SQL

fără mistere

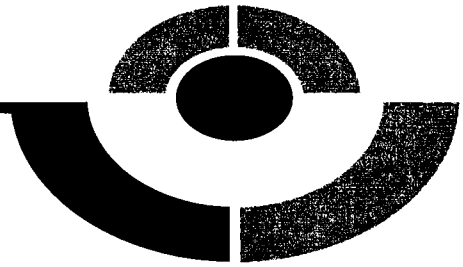
GHID PENTRU AUTODIDACȚI

- Nu este necesară o pregătire formală în SQL!
- O MULȚIME de exemple, care ilustrează APLICAREA CONCEPTELOR
- Concis și cuprinzător, cu un JARGON TEHNIC MINIMAL
- Cartea conține ÎNTREBĂRI RECAPITULATIVE la finalul fiecărui capitol, precum și un EXAMEN FINAL

ROSETTI EDUCATIONAL

Andy Oppel

CUPRINS LA PRIMA VEDERE



CAPITOLUL 1	Concepte ale bazelor de date relaționale	1
CAPITOLUL 2	Concepte SQL	37
CAPITOLUL 3	Definirea obiectelor bazei de date folosind limbajul SQL	53
CAPITOLUL 4	Obținerea datelor folosind limbajul de interogare a datelor (DQL)	87
CAPITOLUL 5	Combinarea datelor din mai multe tabele	125
CAPITOLUL 6	Scrierea interogărilor avansate	149
CAPITOLUL 7	Întreținerea datelor folosind instrucțiuni DML	173
CAPITOLUL 8	Aplicarea controalelor de securitate folosind DCL	185
CAPITOLUL 9	Menținerea integrității bazei de date cu ajutorul tranzacțiilor	205
CAPITOLUL 10	Integrarea limbajului SQL în aplicații	221
CAPITOLUL 11	Elemente de ajustare și performanță ale limbajului SQL	239
	Examen final	261
	Răspunsurile la chestionare și la examenul final	281
	Index	301

CUPRINS

	Mulțumiri	xv
	Introducere	xvii
CAPITOLUL 1	**Concepte ale bazelor de date relaționale**	**1**
	Ce este o bază de date?	2
	Ce este un sistem de gestionare a bazei de date (DBMS) ?	2
	Ce este o bază de date relațională?	3
	Componente ale bazelor de date relaționale	3
	Cum sunt proiectate bazele de date relaționale	9
	Necesitatea normalizării	10
	Aplicarea procesului de normalizare	11
	Prezentare generală a bazei de date pentru un magazin de produse video	21
	Descărcarea codului SQL pentru baza de date folosită ca exemplu	29
	Întrebări recapitulative	31
CAPITOLUL 2	**Concepte SQL**	**37**
	Ce este SQL?	38
	Conectarea la baza de date	38
	Un scurt istoric al limbajului SQL	42
	Convenții de sintaxă SQL	44
	Categorii de instrucțiuni SQL	47
	Limbajul de definire a datelor (DDL)	47

Limbajul de interogare a datelor (DQL) 47
Limbajul de manipulare a datelor (DML) 47
Limbajul pentru controlul datelor (DCL) 48
Comenzile pentru controlul tranzacțiilor 48
Întrebări recapitulative 48

CAPITOLUL 3 **Definirea obiectelor bazei de date folosind limbajul SQL** **53**
Convenții de sintaxă folosite în acest capitol 54
Tipuri de date 54
Tipuri de date predefinite din standardul SQL:2003 55
Extensii ale producătorilor pentru tipuri de date și diferențe față de standard 60
Valori NULL și logica bazată pe trei valori 69
Instrucțiuni DDL (Data Definition Language) 70
Instrucțiunea CREATE DATABASE 70
Instrucțiunea CREATE TABLE 70
Instrucțiunea CREATE INDEX 76
Instrucțiunea CREATE VIEW 77
Instrucțiunea ALTER TABLE 78
Instrucțiunea DROP 80
Întrebări recapitulative 81

CAPITOLUL 4 **Obținerea datelor folosind limbajul de interogare a datelor (DQL)** **87**
Instrucțiunea SELECT de bază
Pseudonime pentru numele coloanelor 90
Sortarea rezultatelor 90
Utilizarea clauzei WHERE pentru filtrarea rezultatelor 94
Operatori de comparare 94
Operatori conjunctivi 97
Operatori logici 100
Operatori aritmetici 108
Funcții SQL elementare 110
Funcții pentru caractere 110
Funcții matematice 115
Funcții de conversie 116

Funcții de agregare și gruparea rândurilor 117
Clauza GROUP BY 119
Operatori pentru interogări compuse 120
UNION 120
UNION ALL 120
INTERSECT 121
EXCEPT 121
Întrebări recapitulative 121

CAPITOLUL 5 **Combinarea datelor din mai multe tabele** **125**
Uniuni (join) 127
Uniuni de egalitate (equijoin) 129
Uniuni naturale 133
Uniuni externe (outer joins) 134
Auto-uniuni (self joins) 138
Alte tipuri de uniuni 140
Uniuni încrucișate (cross joins) 140
Subinterogări 141
Subinterogări necorelate 141
Subinterogări corelate 143
Vizualizări în linie 144
Întrebări recapitulative 144

CAPITOLUL 6 **Scrierea interogărilor avansate** **149**
Funcții SQL avansate 150
Funcții pentru caractere 150
Funcții matematice 154
Funcții pentru dată și oră 156
Folosirea avantajelor oferite de vizualizări 161
Folosirea instrucțiunilor SQL pentru generarea altor instrucțiuni SQL 165
Generarea instrucțiunilor SQL în Oracle 165
Generarea instrucțiunilor SQL în Microsoft SQL Server
Expresia CASE 166
Expresia CASE simplă 166
Expresia CASE cu căutare 168
Întrebări recapitulative 169

CAPITOLUL 7 **Întreținerea datelor folosind instrucțiuni DML** **173**
Instrucțiunea INSERT 175
Inserarea unui singur rând de date folosind clauza VALUES 176
Inserări masive folosind o instrucțiune SELECT internă 177
Instrucțiunea UPDATE 178
Instrucțiunea DELETE 179
Întrebări recapitulative 181

CAPITOLUL 8 **Aplicarea controalelor de securitate folosind DCL** **185**
De ce este necesară securitatea? 186
Arhitecturi de securitate pentru o bază de date 187
Securitatea bazelor de date Microsoft SQL Server și Sybase Adaptive Server 188
Securitatea bazelor de date Oracle 190
Implementarea securității accesului la baza de date 192
Privilegiile bazei de date 192
Instrucțiuni SQL folosite pentru administrarea securității 194
Conturile deținătorilor de scheme 196
Simplificarea administrării folosind roluri 197
Administrarea rolurilor în Microsoft SQL Server și Sybase Adaptive Server 198
Administrarea rolurilor în Oracle 199
Utilizarea vizualizărilor pentru implementarea securității la nivel de coloană și la nivel de rând 200
Întrebări recapitulative 201

CAPITOLUL 9 **Menținerea integrității bazei de date cu ajutorul tranzacțiilor** **205**
Ce este o tranzacție cu baza de date? 206
Suportul pentru tranzacții în sistemele DBMS relaționale 206
Suportul pentru tranzacții în Microsoft SQL Server 207
Suportul pentru tranzacții în Sybase Adaptive Server 208

Suportul pentru tranzacții în Oracle 209
Suportul pentru tranzacții în MySQL 209
Suportul pentru tranzacții în DB2 UDB 210
Blocarea și interblocarea tranzacțiilor 211
Problema actualizării concurente 211
Mecanismele de blocare 212
Interblocările 214
Întrebări recapitulative 216

CAPITOLUL 10 **Integrarea limbajului SQL în aplicații 221**
Procesarea cursorului 222
Instrucțiunea DECLARE CURSOR 222
Instrucțiunea OPEN CURSOR 223
Instrucțiunea FETCH 224
Instrucțiunile UPDATE și DELETE 225
Instrucțiunea CLOSE 225
Înglobarea limbajului SQL în programele de aplicații 226
Conexiunile ODBC 226
Conectarea bazelor de date la aplicațiile Java 227
.NET Framework 228
Limbajul SQL complet computațional 229
Transact-SQL (Microsoft SQL Server și Sybase Adaptive Server) 230
Oracle PL/SQL 232
Întrebări recapitulative 233

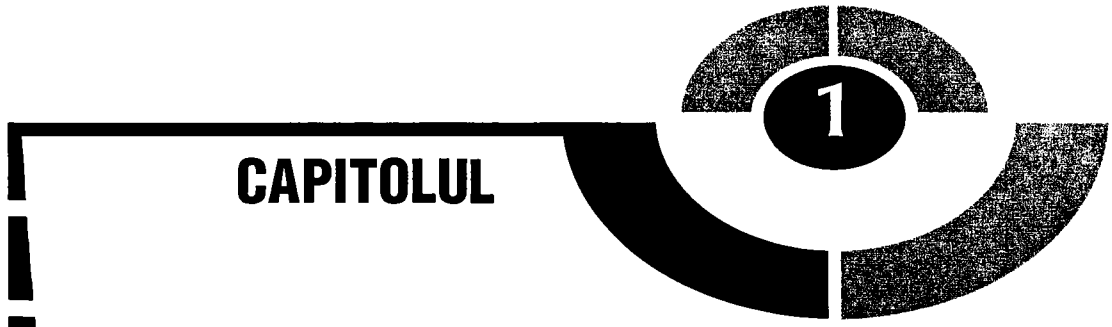
CAPITOLUL 11 **Elemente de ajustare și performanță ale limbajului SQL 239**
Considerații generale de ajustare a sistemelor RDBMS 240
Minimizarea numărului de scrieri și citiri pe disc 240
Ajustarea sistemului de calcul și a mediului de lucru 241
Proiectarea eficientă a tabelelor 241
Optimizarea interogărilor SQL 242
Aspecte generale privitoare la sistemele RDBMS 243
Aspecte privind MySQL 248

Aspecte privind sistemul Oracle 248
Aspecte privind Microsoft SQL Server 252
Aspecte privind sistemul DB2 UDB 253
Optimizarea instrucțiunilor DML 254
Întrebări recapitulative 256

Examen final 261

Răspunsurile la chestionare și la examenul final 281
Capitolul 1 282
Capitolul 2 282
Capitolul 3 283
Capitolul 4 284
Capitolul 5 286
Capitolul 6 288
Capitolul 7 291
Capitolul 8 292
Capitolul 9 293
Capitolul 10 293
Capitolul 11 294
Răspunsurile la examenul final 295

Index 301



Concepte ale bazelor de date relaționale

SQL este principalul limbaj folosit pentru comunicarea cu bazele de date relaționale. Ca urmare, este esențial să înțelegeți conceptele fundamentale ale bazelor de date relaționale înainte de a încerca să învățați limbajul SQL. Acest capitol reprezintă o prezentare generală a conceptelor bazelor de date relaționale. Dacă aceste informații vi se par interesante, vă recomand să consultați și cealaltă carte scrisă de mine, *Databases Demystified* (McGraw-Hill/Osborne, 2004), care se ocupă în întregime de proiectarea, utilizarea și gestionarea bazelor de date relaționale.

Ce este o bază de date?

O *bază de date* este o colecție de informații interrelaționate gestionate ca o singură unitate. Această definiție este intenționat foarte largă, deoarece există mari diferențe între concepțiile diferiților producători care pun la dispoziție sisteme de baze de date. De exemplu, Oracle Corporation definește o bază de date ca fiind o colecție de fișiere fizice gestionate de o singură instanță (copie) a produsului software pentru baze de date, în timp ce Microsoft definește o bază de date SQL Server ca fiind o colecție de date și alte obiecte. Un *obiect* al bazei de date este o structură de date denumită stocată în baza de date, cum ar fi un tabel, o vizualizare sau un index. Veți afla mai multe despre obiectele bazelor de date în secțiunea „Componente ale bazelor de date relaționale", mai târziu în acest capitol.

Există mari diferențe între implementările furnizorilor de baze de date. În majoritatea sistemelor de baze de date, datele sunt stocate în mai multe fișiere fizice, dar în Microsoft Access toate obiectele bazei de date, împreună cu datele care aparțin unei baze de date, sunt stocate într-un singur fișier fizic. (Un *fișier* este o colecție de înregistrări înrudite stocate ca o singură unitate de sistemul de operare al calculatorului.) Alte baze de date relaționale, în special în implementările mai vechi, stochează fiecare element al bazei de date într-un fișier separat. Totuși, unul dintre principalele avantaje ale bazelor de date relaționale este faptul că detaliile de implementare fizică sunt separate de definițiile logice ale obiectelor bazei de date, astfel încât majoritatea utilizatorilor bazei de date nu au nevoie să știe unde (și cum) sunt stocate obiectele bazei de date în sistemul de fișiere al calculatorului. De fapt, pe măsură ce veți învăța limbajul SQL, veți vedea că nu este nevoie să specificați numele unui fișier fizic într-o instrucțiune SQL decât atunci când definiți sau modificați chiar obiectele bazei de date – *niciodată* nu este necesară specificarea unui fișier atunci când adăugați, modificați, ștergeți sau obțineți date stocate în obiectele bazei de date.

Ce este un sistem de gestionare a bazei de date (DBMS) ?

Un *sistem de gestionare a bazei de date (DBMS – database management system)* este un produs software furnizat de producătorul bazei de date. Produse software precum Microsoft Access, Microsoft SQL Server, Oracle Database, Sybase, DB2, INGRES, MySQL și PostgreSQL fac parte din categoria DBMS sau, mai corect, *DBMS relaționale (RDBMS)*. Bazele de date relaționale sunt definite și prezentate în secțiunea următoare a acestui capitol.

Sistemul DBMS pune la dispoziție toate serviciile de bază necesare pentru organizarea și întreținerea bazei de date, inclusiv următoarele:

- Transferarea datelor în și din fișierele fizice de date, în funcție de cerințe.
- Gestionarea accesului concurențial la date al mai multor utilizatori, inclusiv prevenirea conflictelor care ar putea fi cauzate de actualizările simultane.
- Gestionarea tranzacțiilor, astfel încât toate modificările făcute asupra bazei de date printr-o tranzacție să fie executate ca o singură unitate. Cu alte cuvinte, dacă tranzacția reușește, toate modificările efectuate de tranzacție sunt înregistrate în baza de date; dacă tranzacția eșuează, nici una dintre modificări nu este înregistrată în baza de date. Totuși, rețineți că unele sisteme RDBMS nu asigură suportul pentru tranzacții.
- Acceptă un *limbaj de interogare*, care reprezintă sistemul de comenzi folosit de utilizator pentru a obține date din baza de date. SQL este principalul limbaj folosit pentru sistemele DBMS relaționale și subiectul principal al acestei cărți.
- Funcții pentru salvarea bazei de date și pentru refacerea bazei de date în urma erorilor.
- Mecanisme de securitate pentru împiedicarea accesului neautorizat la date și modificarea acestora.

Ce este o bază de date relațională?

O *bază de date relațională* este o bază de date care respectă modelul relațional, dezvoltat de Dr.E.F.Codd. Modelul relațional prezintă datele sub forma familiarelor tabele bidimensionale, similar cu o foaie de calcul tabelar. Spre deosebire de o foaie de calcul tabelar, nu este obligatoriu ca datele să fie stocate într-o formă tabelară, iar modelul permite și combinarea tabelelor (*crearea uniunilor (joining)*, în terminologia relațională) pentru formarea vizualizărilor, care sunt prezentate tot ca tabele bidimensionale. Flexibilitatea extraordinară a bazelor de date relaționale este dată de posibilitatea de a folosi tabelele independent sau în combinații, fără nici o ierarhie sau secvență predefinită în care trebuie să se facă accesul la date.

Componente ale bazelor de date relaționale

Să aruncăm o privire asupra componentelor fundamentale ale bazelor de date relaționale. Aceste componente sunt utilizate pentru a construi obiectele bazelor de date pe care le vom folosit în cartea de față. Instrucțiunile SQL folosite pentru crearea acestor componente ale bazei de date sunt prezentate în capitolul 3.

Tabele

Unitatea primară de stocare a datelor într-o bază de date relațională este *tabelul*, care este o structură bidimensională compusă din rânduri și coloane. Fiecare tabel reprezintă o *entitate*, ceea ce înseamnă o persoană, un loc, un lucru sau un eveniment care trebuie să fie reprezentat în baza de date, cum ar fi un client, un cont bancar sau o tranzacție bancară. Fiecare rând al tabelului reprezintă o apariție a entității. Figura 1-1 reprezintă listingul parțial al unui tabel numit MOVIE (filme).

Tabelul MOVIE este parte a unei baze de date pentru un magazin de produse video, folosit ca exemplu în toată această carte. Restul bazei de date folosite ca exemplu este prezentată în secțiunea „Prezentare generală a bazei de date pentru un magazin de produse video", către sfârșitul acestui capitol. Tabelul MOVIE conține date care descriu filmele disponibile în magazinul de produse video. Fiecare rând din tabel reprezintă un film, iar fiecare coloană reprezintă o caracteristică a filmului respectiv, cum ar fi titlul filmului sau categoria MPAA*.

MOVIE_ID	MOVIE_GENRE_CODE	MPAA_RATING_CODE	MOVIE_TITLE	RETAIL_PRICE_VHS	RETAIL_PRICE_DVD	YEAR_PRODUCED
1	Drama	R	Mystic River	58.97	19.96	2003
2	ActAd	R	The Last Samurai	15.95	19.96	2003
3	Comdy	PG-13	Something's Gotta Give	14.95	29.99	2003
4	ActAd	PG-13	The Italian Job	11.95	19.99	2003
5	ActAd	R	Kill Bill: Vol. 1	24.99	29.99	2003
6	ActAd	PG-13	Pirates of the Caribbean: The Curse of the Black Pearl	24.99	29.99	2003
7	Drama	PG-13	Big Fish	14.95	19.94	2003
8	ActAd	R	Man on Fire	50.99	29.98	2004
9	ActAd	PG-13	Master and Commander: The Far Side of the World	12.98	39.99	2003
10	Drama	R	Lost in Translation	49.99	14.98	2003
11	Rmce	PG-13	Two Weeks Notice	6.93	14.97	2002
12	Comdy	PG-13	50 First Dates	9.95	19.94	2004
13	Comdy	PG-13	Matchstick Men	6.93	19.97	2003
14	Drama	R	Cold Mountain	24.99	29.99	2003
15	Drama	R	Road to Perdition	9.99	14.99	2002
16	Comdy	PG-13	The School of Rock	11.69	29.99	2003
17	Rmce	PG-13	13 Going on 30	14.94	28.95	2004
18	Drama	R	Monster	24.99	29.99	2003
19	ActAd	PG-13	The Day After Tomorrow	12.98	29.98	2004
20	Forgn	R	Das Boot	17.99	19.94	1981

Figura 1-1 Listingul tabelului MOVIE

* Motion Picture Association of America (MPAA) a fost fondată în Statele Unite, în 1922, ca asociație a producătorilor de filme, cu scopul de a îmbunătăți imaginea pe care o avea publicul despre industria cinematografică. (nota trad.)

Probabil ați observat asemănarea dintre tabelele bazelor de date relaționale și foile de calcul tabelar. Totuși, așa cum veți vedea în restul acestui capitol, bazele de date relaționale vă oferă o flexibilitate mult mai mare în organizarea și afișarea informațiilor.

Relații

Relațiile reprezintă asocierile dintre tabelele bazelor de date relaționale. Deși fiecare tabel relațional poate exista independent, esența bazelor de date este tocmai stocarea informațiilor între care există legături. De exemplu, pe lângă filmele propriu-zise, puteți stoca și informații despre categoriile folosite de magazin pentru organizarea inventarelor de filme. În același timp, puteți stoca și informații despre copiile fiecărui film, inclusiv data la care a fost primită copia și formatul acesteia, cum ar fi DVD sau VHS. Prin folosirea relațiilor, puteți asocia tabelele înrudite, într-un mod formal, ușor de folosit astfel încât vreți să combinați date din tabele multiple în aceeași interogare a bazei de date, dar păstrând flexibilitatea de a include numai informațiile care vă interesează. Posibilitatea de a selecta din baza de date numai informațiile care vă interesează vă permite să ajustați informațiile din baza de date în funcție de cerințele specifice ale persoanelor sau aplicațiilor care au acces la baza de date.

Figura 1-2 prezintă patru tabele din baza de date a magazinului de produse video și relațiile dintre acestea, într-un format cunoscut sub numele de diagramă de relații a entităților (ERD – Entity Relationship Diagram). Diagramele ERD vă pun la dispoziție o modalitate de prezentare a proiectului general al unei baze de date relaționale, într-un format ușor de înțeles pentru utilizatorii bazei de date, indiferent dacă au sau nu cunoștințe tehnice. Fiecare dreptunghi din diagramă reprezintă un tabel relațional, cu numele tabelului scris deasupra liniei orizontale și coloanele tabelului enumerate pe verticală, în porțiunea principală a dreptunghiului. Dacă doriți, puteți să comparați tabelul MOVIE din figura 1-2 cu listingul aceluiași tabel, din figura 1-1, pentru a vă ajuta să vizualizați conținutul tabelului.

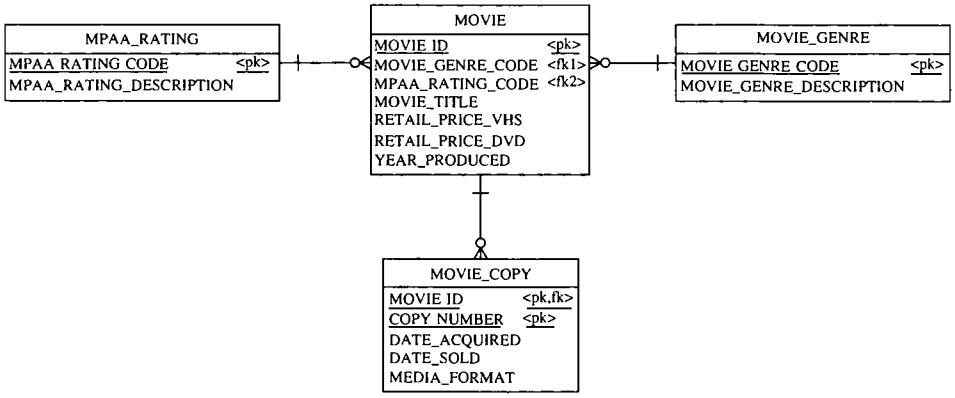


Figura 1-2 Diagrama ERD a bazei de date pentru magazinul de produse video (prezentare parțială)

Fiecare relație este prezentată în diagrama ERD ca o linie ce conectează două tabele. Cele două capete ale liniei arată *cardinalitatea maximă* a relației, respectiv numărul maxim de rânduri dintr-un tabel care pot fi asociate cu un rând dat din tabelul aflat la celălalt capăt al relației. Cardinalitatea maximă poate fi *unu* (caz în care linia nu are nici un simbol special la capăt) sau *mai multe* (caz în care linia se termină cu un simbol numit *picior de cioară (crow's foot)* – linia se împarte la capăt în trei segmente). În apropiere de capătul liniei se află un alt simbol, care arată *cardinalitatea minimă**, adică numărul minim de rânduri dintr-un tabel care poate fi asociat cu tabelul de la celălalt capăt al relației. Cardinalitatea minimă poate fi *zero*, indicată printr-un cerc desenat pe linie, sau *unu*, indicată printr-o liniuță care taie linia relației. De exemplu, relația dintre tabelele MPAA_RATING și MOVIE din figura 1-2 este o relație de tip *unu-la-mai-mulți*, ceea ce înseamnă că fiecare rând din tabelul MPAA_RATING (tabelul din partea „unu", numit și tabel *părinte***) poate fi asociat cu mai multe rânduri din tabelul MOVIE (tabelul din partea „mai mulți" a relației, numit și tabel *copil****), dar fiecare rând din tabelul MOVIE poate fi asociat cu un singur rând din tabelul MPAA_RATING. Relația este logică, deoarece orice film lansat în SUA poate avea o singură categorie MPAA, dar o categorie poate fi asociată mai multor filme diferite. Este adevărat că filmele sunt uneori „cenzurate" pentru a putea fi încadrate în diferite categorii, dar această problemă este rezolvată ușor, prin tratarea diferitelor versiuni ale aceluiași film ca și cum ar fi filme diferite, la fel cum facem atunci când un film este reluat cu alți actori. Este foarte important să țineți seama de aceste lucruri, deoarece bazele de date relaționale acceptă numai relațiile de tip unu-la-mai-mulți.

Cardinalitatea minimă arată dacă participarea la o anumită relație este obligatorie sau opțională. Toate relațiile din figura 1-2 sunt obligatorii în partea „unu" și opționale în partea „mai mulți", aceasta fiind cea mai frecvent folosită formă de relație. Dacă ne uităm din nou la relația dintre tabelele MPAA_RATING și MOVIE, aceasta înseamnă că fiecare rând din tabelul MOVIE *trebuie* să aibă un rând corespondent în tabelul MPAA_RATING, dar nu este obligatoriu ca fiecare rând din tabelul MPAA_RATING să aibă asociat un rând din tabelul MOVIE. Dacă vreți să permiteți ca inventarul de filme al magazinului să conțină titluri care nu au asociată o categorie MPAA, liniuța de la capătul dinspre tabelul MPAA_RATING al liniei care reprezintă relația cu tabelul MOVIE va fi înlocuită de un cerc. Deși sunt relativ frecvent întâlnite cazurile în care partea „unu" a unei relații nu este obligatorie, este foarte neobișnuit să aveți o relație în care să fie obligatorie partea „mai mulți" a relației, ceea ce ar însemna că tabelul părinte trebuie să aibă în orice moment cel puțin un „copil" în baza de date. Gândiți-vă ce s-ar întâmpla dacă ați face obligatorie partea dinspre tabelul MOVIE a relației cu tabelul MPAA_RATING. Dacă asociația MPAA

---

* numită și *ordinalitate* (nota trad.)

** numit și tabel *determinant*. În această traducere, vom folosi în continuare forma *tabel părinte*, mai apropiată de termenul folosit de autor (nota trad.)

*** numit, în funcție de sursa documentară, și tabel *dependent*, tabel *descendent* sau tabel *subordonat*. În această traducere, vom folosi în continuare forma *tabel copil*, mai apropiată de termenul folosit de autor (nota trad.)

(Motion Picture Association of America) ar inventa un nou cod de clasificare a filmelor, nu ați putea să adăugați acest cod în tabelul MPAA_RATING decât dacă ați avea și un film din categoria respectivă în tabelul MOVIE. La fel, nu ați putea să ștergeți din tabelul MOVIE ultimul rând corespunzător unei categorii decât dacă ați șterge și categoria respectivă din tabelul MPAA_RATING. Aceste restricții greoaie sunt probabil motivul pentru care bazele de date relaționale nu asigură suportul direct pentru existența „copiilor" obligatorii în relațiile de tip „unu-la-mai-mulți".

Relațiile sunt implementate folosind coloane corespondente din cele două tabele participante. În diagrama ERD, coloana sau coloanele subliniate din fiecare tabel, având în dreapta notația „pk", reprezintă cheia primară (primary key), adică o coloană sau un set de coloane care identifică în mod unic fiecare rând dintr-un tabel. Un tabel poate avea o singură cheie primară. Totuși, o cheie primară poate fi compusă din mai multe coloane, dacă aceasta este calea de formare a unei chei unice. Dacă o cheie primară este folosită într-un alt tabel pentru stabilirea unei relații, poartă numele de cheie externă. În figura 1-2, observați coloanele cheie externă folosite în tabelul MOVIE pentru crearea relațiilor cu tabelele MOVIE_GENRE și MPAA_RATING și marcate cu identificatoarele „<fk1>" și „<fk2>" în dreapta numelui coloanei cheie externă. Și coloana LANGUAGE_CODE este marcată drept cheie externă (cu identificatorul „<fk3>"), dar tabelul LANGUAGE și relația acestuia cu tabelul MOVIE au fost omise din figura 1-2. de asemenea, observați că cheia primară a tabelului MOVIE apare în tabelul „copil" MOVIE_COPY cu rolul de cheie externă, pentru a stabili o relație între cele două tabele.

Cheile primare și cheile externe sunt blocuri de construcție fundamentale ale modelului relațional, deoarece stabilesc relații și permit crearea legăturilor între date, atunci când este necesar. Trebuie să înțelegeți acest concept pentru a putea înțelege cum funcționează bazele de date relaționale.

Restricții

O restricție* este o regulă specificată pentru un obiect al bazei de date (de obicei, un tabel sau o coloană), având rolul de a limita într-un mod oarecare domeniul de valori permise pentru obiectul respectiv al bazei de date. După ce sunt specificate, restricțiile sunt impuse automat de sistemul DBMS și nu pot fi ocolite decât dacă o persoană autorizată le dezactivează sau le șterge (le elimină). Fiecare restricție primește un nume unic, astfel încât să poată fi referită în mesajele de eroare și în comenzile folosite ulterior în baza de date. Este recomandabil ca proiectanții bazei de date să denumească restricțiile, deoarece numele generate automat de baza de date nu sunt foarte descriptive. Totuși, nu am denumit restricțiile din baza de date folosită ca exemplu în această carte, deoarece, din păcate, nu toate produsele RDBMS disponibile în prezent acceptă restricțiile denumite.

* *constraint*, în limba engleză. Numită uneori *constrângere*. (nota trad.)

Există mai multe tipuri de restricții pentru baze de date:

- ***Restricția NOT NULL*** Poate fi plasată pe o coloană pentru a împiedica folosirea valorilor nule. O valoare nulă (null) este o modalitate specială prin care sistemul RDBMS tratează valoarea unei coloane pentru a indica faptul că valoarea coloanei respective nu este cunoscută. O valoare nulă nu este același lucru cu un spațiu liber, un șir vid sau valoarea zero – este o valoare specială care nu este egală cu nimic altceva. Veți afla mai multe despre valorile nule în capitolul 3.
- ***Restricția cheie primară (primary key)*** Definită pe coloana (coloanele) cheie primară ale unui tabel pentru a garanta că valorile cheie primară sunt întotdeauna unice în întreg tabelul. Atunci când cheia primară este definită pe mai multe coloane, combinația valorilor acelor coloane trebuie să fie unică în tabel – o coloană care reprezintă doar o parte a cheii primare poate conține valori duplicate în tabel. Restricțiile cheie primară sunt aproape întotdeauna implementate de RDBMS prin folosirea unui index. Indexul este un tip special de obiect al bazei de date care permite efectuarea căutărilor rapide în valorile coloanei. Atunci când în tabele sunt inserate rânduri noi, sistemul RDBMS verifică automat indexul pentru a se asigura că cheia primară a noului rând nu este deja folosită în tabel și, dacă se întâmplă acest lucru, respinge cererea de inserare. Căutarea în indexuri se face mult mai repede decât căutarea în tabel; ca urmare, indexarea cheii primare este esențială pentru orice tabel, indiferent de dimensiunea acestuia, astfel încât căutarea cheilor duplicate la fiecare inserare să nu ducă la o reducere semnificativă a performanțelor. O caracteristică suplimentară a cheilor primare este faptul că nu pot fi definite decât pe coloane pentru care a fost definită și restricția NOT NULL.
- ***Restricția de unicitate (unique)*** Definită pe o coloană sau un set de coloane care trebuie să conțină valori unice în cadrul tabelului. Ca și în cazul cheilor primare, sistemul RDBMS folosește aproape întotdeauna un index ca modalitate de impunere eficientă a restricției. Totuși, spre deosebire de cheile primare, un tabel poate avea definite mai multe restricții de unicitate, iar coloanele care participă la o restricție de unicitate pot conține (în cele mai multe sisteme RDBMS) și valori nule.
- ***Restricția referențială*** **(numită uneori *restricție de integritate referențială*)** O restricție care impune o relație între două tabele dintr-o bază de date relațională. Prin „impunere" se înțelege că sistemul RDBMS se asigură întotdeauna, în mod automat, că fiecărei valori a cheii externe îi corespunde o valoare a cheii primare în tabelul părinte. În tabelul MOVIE (a se vedea figura 1-1), sistemul RDBMS nu-mi permite să inserez o înregistrare cu valoarea „M" în coloana MPAA_RATING_CODE, deoarece „M" nu este o valoare MPAA_RATING_CODE validă și, ca urmare, nu apare ca valoare a cheii

primare din tabelul MPAA_RATING. În sens invers, sistemul RDBMS nu-mi permite să șterg din tabelul MPAA_RATING rândul cu cheia primară „PG-13", deoarece valoarea respectivă este folosită ca valoare a cheii externe pentru cel puțin una din înregistrările din tabelul MOVIE. Pe scurt, restricția referențială garantează că relația dintre cele două tabele și valorile corespondente ale cheii primare și cheii externe își păstrează logica în orice moment.

- ***Restricția CHECK*** Folosește o instrucțiune logică simplă (scrisă în SQL) pentru a valida valoarea unei coloane. Rezultatul instrucțiunii trebuie să fie o valoare logică de adevărat (true) sau fals (false), astfel încât un rezultat adevărat să permită inserarea în tabel a valorii coloanei, iar un rezultat fals să ducă la rejectarea valorii coloanei, cu mesajul de eroare corespunzător.

Vizualizări

O vizualizare (view) este o interogare stocată în baza de date și pune la dispoziția utilizatorului un subset personalizat al datelor din unul sau mai multe tabele ale bazei de date. Cu alte cuvinte, o vizualizare este un tabel virtual, deoarece arată ca un tabel și, în cele mai multe privințe, se comportă ca un tabel, dar nu stochează date (nu este stocată decât interogarea SQL care definește vizualizarea).

Vizualizările au mai multe funcții utile:

- Maschează coloanele pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează rândurile pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează operațiile complexe efectuate în baza de date, cum ar fi uniunile de tabele (respectiv combinarea coloanelor din tabele multiple într-o singură interogare a bazei de date).
- Îmbunătățesc performanțele interogărilor (în unele sisteme RDBMS, precum Microsoft SQL Server).

Cum sunt proiectate bazele de date relaționale

Această secțiune prezintă, foarte pe scurt, procesul general de proiectare a bazelor de date. Atunci când ați văzut pentru prima dată figura 1-2, mai devreme în acest capitol, probabil v-ați întrebat de ce coloanele au fost plasate în tabele multiple și de ce o anumită coloană a fost inclusă într-un anumit tabel și nu în altul. Scopul acestei secțiuni este de a vă ajuta să răspundeți la întrebările de mai sus și de a vă oferi un punct de plecare, dacă decideți să proiectați propriile baze de date pe măsură ce învățați limbajul SQL. Totuși, proiectarea

bazelor de date este un domeniu mult mai larg, suficient pentru o întreagă carte. Dacă vi se pare interesant și vreți să aflați mai multe, veți găsi o mulțime de pagini web în Internet, precum și alte cărți dedicate acestui subiect, inclusiv prima mea carte, *Databases Demystified*.

În 1972, Dr. E. F. Codd, părintele bazelor de date relaționale, și-a dat seama că tabelele relaționale care îndeplinesc anumite criterii pun mai puține probleme la inserarea, actualizarea sau ștergerea datelor. Ca urmare, a pus la punct un set de reguli care trebuie respectate (organizate în trei „forme normale"*) și un proces numit normalizare, care este o tehnică pentru producerea unui set de relații (termenul folosit de Dr. Codd pentru tabele) cu proprietățile dorite.

Necesitatea normalizării

Figura 1-3 prezintă tabelul MOVIE fără normalizare, așa cum ar arăta dacă toate informațiile despre filme ar fi colectate într-un singur tabel. Acest exemplu va fi folosit pentru ilustrarea procesului de normalizare. În general, numele coloanelor din tabelele relaționale folosesc liniuțe de subliniere pentru separarea cuvintelor. În discuția despre normalizare am eliminat aceste liniuțe din figuri, pentru a face textul mai ușor de citit.

Există trei probleme care pot apărea în tabelele fără normalizare din bazele de date relaționale și toate trei există și în tabelul din figura 1-3. Scopul procesului de normalizare este de a elimina aceste probleme (anomalii) din proiectul bazei de date.

MOVIE ID	GENRE CODE	GENRE DESC.	LANG. CODE	MPAA RATING CODE	MPAA RATING DESC.	MOVIE TITLE	YEAR PRODUCED	DATE ACQUIRED	DATE SOLD	MEDIA FORMAT	RETAIL PRICE
1	Drama	Drama	en, fr	R	Under 17 requires accompanying parent or adult guardian	Mystic River	2003	1/1/2005		DVD	19.96
2	ActAd	Action and Adventure	en, fr, es	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		DVD	19.96
2	ActAd	Action and Adventure	en, fr, es	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		VHS	15.95
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	1/10/2005	1/30/2005	DVD	29.99
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	2/15/2005		DVD	29.99
4	ActAd	Action and Adventure	en, fr	PG-13	Parents strongly cautioned	The Italian Job	2003	2/15/2005		DVD	19.99

Figura 1-3 Tabelul MOVIE fără normalizare

* *normal forms* în limba engleză. Numite și *forme normalizate*. (nota trad.)

Anomalia de inserare

Anomalia de inserare se referă la o situație în care nu puteți insera date în baza de date din cauza unei dependențe artificiale dintre coloanele unui tabel. Să presupunem că vreți să adăugați în baza de date a magazinului de produse video un nou gen de film (în coloanele GENRE_CODE și GENRE_DESCRIPTION) care urmează a fi folosit pentru clasificarea filmelor. Modelul din figura 1-3 nu permite acest lucru decât dacă aveți un film care să fie plasat în categoria respectivă și pe care va trebui să-l adăugați în tabelul MOVIE în același timp. Aceeași restricție este valabilă și pentru coloanele MPAA_RATING_CODE și MPAA_RATING_DESCRIPTION. Ar fi mult mai bine dacă ați putea adăuga noile genuri și categorii înainte de primirea filmelor în magazin.

Anomalia de ștergere

Anomalia de ștergere este inversul anomaliei de inserare. Se referă la situația în care ștergerea unor date duce la pierderea neintenționată a altor date. De exemplu, dacă primul film din figura 1-3 (*Mystic River*) este singurul rând din tabelul MOVIE pentru care coloana GENRE_CODE are valoarea „Drama" și este șters, se pierde informația că a existat vreodată un gen numit „Drama". Același lucru este adevărat și dacă ștergeți ultimul film din tabelul MOVIE pentru care coloana MPAA_RATING_CODE are o anumită valoare.

Anomalia de actualizare

Anomalia de actualizare se referă la o situație în care actualizarea unei singure valori necesită actualizarea mai multor rânduri. De exemplu, dacă în tabelul prezentat în figura 1-3 trebuie să modificați descrierea codului MPAA_RATING_CODE „R", trebuie să modificați și toate rândurile din tabel pentru filmele cu codul respectiv. Probleme similare apar și pentru coloana GENRE_DESCRIPTION. Chiar și coloana RETAIL_PRICE are această problemă, deoarece toate copiile aceluiași film (cu aceeași valoare MOVIE_ID) pe același mediu (DVD sau VHS) ar trebui să aibă același preț. Un alt pericol legat de această anomalie este faptul că stocarea unor date redundante poate duce la posibilitatea de a actualiza numai o parte a copiilor respectivelor date, ceea ce ar avea ca rezultat apariția inconsecvențelor în baza de date.

Aplicarea procesului de normalizare

De obicei, normalizarea începe de la mijloacele de redare a datelor care sunt (sau vor fi) prezentate utilizatorilor, cum ar fi pagini web, ecrane ale aplicațiilor, rapoarte și așa mai departe. Colectiv, acestea sunt numite *vizualizări de utilizator (user views)*. Poate părea ciudat la prima vedere, dar este ceva obișnuit ca proiectarea unui sistem de prelucrare a datelor să înceapă de la rezultatele pe care le va vedea utilizatorul, parcurgând apoi drumul înapoi către mijloacele folosite pentru obținerea rezultatelor dorite. În timpul proiectării

bazei de date, procesul de normalizare este aplicat fiecărei vizualizări, iar rezultatul este un set de relații normalizate care pot fi apoi direct implementate ca tabele ale bazei de date relaționale. Procesul în sine este destul de simplu, iar regulile nu sunt foarte dificile. Totuși, stăpânirea procesului de normalizare cere timp și exercițiu, în special deoarece impune proiectantului să se gândească într-un mod conceptual la datele și relațiile pe care intenționează să le folosească. În timpul normalizării, considerați că fiecare vizualizare este o relație. Cu alte cuvinte, conceptualizați fiecare vizualizare ca și cum ar fi deja un tabel bidimensional – un mod de lucru pentru care aveți nevoie de experiență.

De asemenea, este nevoie de timp pentru a vă obișnui cu terminologia folosită în procesul de normalizare. În timpul acestui proces, majoritatea proiectanților evită folosirea unor termeni fizici, precum tabel, coloană și cheie primară. Deși relația pe cale de a fi normalizată reprezintă o propunere de tabel, încă nu există ca tabel fizic, așa că termenul nu este foarte exact. Folosim termenul de *relație* în loc de *tabel*, *atribut* în loc de *coloană* și *identificator unic* în loc de *cheie primară*. Pentru începători, folosirea termenilor fizici pare mai naturală, dar trebuie să cunoașteți terminologia preferată atunci când căutați informații sau exemple din alte surse. Deși, în majoritatea sistemelor RDBMS, la denumirea obiectelor nu se face diferențierea literelor mari de cele mici, am scris toate numele de coloane cu majuscule, pentru consecvență. Totuși, pentru numele atributelor și ale relațiilor am folosit combinații de litere mari și mici, deoarece aceasta este practica obișnuită în domeniu.

Procesul de normalizare este aplicat sistematic fiecărei vizualizări. Cel puțin la început, este mai ușor să reprezentați fiecare vizualizare ca un tabel bidimensional, conținând date reprezentative, așa cum am făcut în figura 1-3. Pe măsură ce parcurgeți procesul de normalizare, veți rescrie relațiile existente și veți crea altele. Rescrierea vizualizărilor în relații (tabele) cu date reprezentative este un proces obositor și consumator de timp. Trebuie să fiți foarte atent ca exemplele de date folosite pentru luarea deciziilor în procesul de normalizare să fie cu adevărat reprezentative pentru tipurile de valori care vor apărea în datele reale. Așa cum probabil vă așteptați, exemple prost alese duc deseori la proiectarea eronată a bazei de date. Vestea bună este că, în timp, vă veți obișnui să vizualizați exemplele de date, fără să mai fie necesar să le înregistrați pe toate.

Rețineți că scopul procesului de normalizare este eliminarea anomaliilor de inserare, actualizare și ștergere. Procesul determină crearea unui număr mai mare de relații decât ați avea într-un model fără normalizare. Relațiile suplimentare sunt necesare pentru eliminarea anomaliilor, dar împărțirea datelor în mai multe relații face ca extragerea datelor stocate să fie puțin mai dificilă. De fapt, sacrificați o parte din performanțele de extragere a datelor și din ușurința utilizării pentru ca operațiile de inserare, actualizare și ștergere să fie mai simple.

Alegerea unui identificator unic

Primul pas al procesului de normalizare constă în alegerea unui *identificator unic (unique identifier)*, care este un atribut (o coloană) sau un set de atribute care identifică în mod unic fiecare rând de date dintr-o relație.

Identificatorul unic va deveni ulterior cheia primară a tabelului creat din relația normalizată. Pentru normalizare, este *obligatoriu* ca fiecare relație să aibă un identificator unic. În multe cazuri, puteți găsi un atribut care identifică în mod unic datele din fiecare rând al relației pe care vreți să o normalizați. Atunci când nu puteți găsi un singur atribut care să poată fi folosit ca identificator unic, este posibil să găsiți mai multe atribute care pot fi concatenate (combinate) pentru a forma un identificator unic. Atunci când identificatoarele unice sunt formate din atribute multiple, fiecare atribut rămâne pe propria lui coloană – nu faceți decât să definiți un identificator unic format din mai multe coloane. În foarte puține cazuri, într-o relație nu există un set rezonabil de atribute care să poată fi folosit ca identificator unic. Atunci când se întâmplă acest lucru, trebuie să inventați un identificator unic, deseori cu valori atribuite secvențial sau aleatoriu pe măsură ce noile rânduri de date sunt adăugate în tabelul bazei de date. Această tehnică este sursa unor identificatoare unice, precum numărul de asigurări sociale folosit în Statele Unite, numerele de identificare ale angajaților sau numerele de înmatriculare ale mașinilor.

Relația Movie din figura 1-3 ne pune o problemă în privința găsirii unui identificator unic. La prima vedere, ar părea că atributul Movie ID este cel mai potrivit în acest scop. Totuși, observați că valorile Movie ID „2" și „3" apar de câte două ori, așa că, fără nici un dubiu, această valoare nu este unică. Problema este că valoarea Movie ID identifică în mod unic fiecare titlu, dar magazinul urmărește separat fiecare copie a filmelor pe care le are în stoc. Cauza este faptul că magazinul se ocupă și de închirierea filmelor și vrea să se asigure că fiecare client returnează exact copia pe care a închiriat-o. După inspectarea datelor folosite ca exemplu și o scurtă discuție cu proprietarul magazinului, ajungeți la concluzia că în relația Movie nu există nici o combinație de atribute care să identifice în mod unic fiecare copie a unui film, așa că inventați un atribut numit Copy Number și-l adăugați în relație. Ori de câte ori inventați un identificator unic (sau o parte a unui identificator) este foarte important ca toți să înțeleagă valorile pe care le va lua acest identificator. În acest caz, proprietarul magazinului decide ca valorile Copy Number să reînceapă de la 1 pentru fiecare valoare Movie ID, ceea ce înseamnă că valorile Copy Number sunt unice numai în combinație cu valorile Movie ID. Relația rezultată este prezentată în figura 1-4.

Prima formă normală: eliminarea datelor repetate

O relație este în *prima formă normală** atunci când nu conține atribute cu valori multiple (atribute multivaloare), adică atribute care au mai multe valori pentru același rând de date. Într-o relație, orice intersecție a unui rând cu o coloană trebuie să conțină *cel mult* o valoare pentru ca relația să fie în prima formă normală. În figura 1-4, atributul pentru limbă (Lang. Code) conține mai multe valori pentru unele dintre filme, așa că-l puteți considera un

* Pentru a fi în prima formă normală (*1NF* sau *first normal form*), toate tabelele trebuie să aibă următoarele două proprietăți: **a)** fiecare coloană trebuie să conțină o singură valoare (descrisă uneori ca fiind *atomică* sau *indivizibilă*) și **b)** nici un tabel nu poate avea coloane repetate pentru date înrudite (nota trad.)

Movie:

MOVIE ID	COPY NUMBER	GENRE CODE	GENRE DESC.	LANG. CODE	MPAA RATING CODE	MPAA RATING DESC.	MOVIE TITLE	YEAR PRODUCED	DATE ACQUIRED	DATE SOLD	MEDIA FORMAT	RETAIL PRICE
1	1	Drama	Drama	en, fr	R	Under 17 requires accompanying parent or adult guardian	Mystic River	2003	1/1/2005		DVD	19.96
2	1	ActAd	Action and Adventure	en, fr, es	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		DVD	19.96
2	2	ActAd	Action and Adventure	en, fr, es	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		VHS	15.95
3	1	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	1/10/2005	1/30/2005	DVD	29.99
3	2	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	2/15/2005		DVD	29.99
4	1	ActAd	Action and Adventure	en, fr	PG-13	Parents strongly cautioned	The Italian Job	2003	2/15/2005		DVD	19.99

Figura 1-4 Relația Movie după adăugarea atributului Copy Number

atribut cu valori multiple. Atributele de acest tip sunt mai greu de întreținut, deoarece valorile din listă trebuie să fie mai întâi separate, astfel încât valorile individuale să poată fi modificate fără a le afecta pe celelalte.

Uneori, un atribut multivaloare este deghizat sub forma atributelor multiple. De exemplu, figura 1-4 ar putea fi modificată astfel încât să conțină atribute (coloane) separate pentru cel mult trei limbi corespunzătoare fiecărui film, numite Language 1, Language 2 și Language 3. Totuși, și acestea ar fi considerate atribute multivaloare, dar într-o formă specială, numită *grup repetitiv*, care nu este acceptat în prima formă normală. Din punct de vedere logic, un grup repetitiv nu este diferit de un atribut multivaloare. De fapt, grupurile repetitive prezintă deseori chiar mai multe probleme decât atributele multivaloare, deoarece trebuie să adăugați o nouă coloană în tabel ori de câte ori vreți să adăugați mai multe valori decât a prevăzut inițial proiectantul bazei de date (cum ar fi o a patra limbă pentru un film). Bazele de date relaționale cer ca toate rândurile dintr-un tabel să aibă același număr de coloane, dar un tabel poate conține orice număr de rânduri. Ca urmare, trucul este să transformați coloanele repetate și valorile repetate din coloane în *rânduri* repetate într-un tabel separat – și exact acest lucru vă cere să-l faceți procesul de normalizare pentru obținerea primei forme normale.

Pentru transformarea relațiilor ne-normalizate în prima formă normală, trebuie să mutați atributele multivaloare și grupurile repetitive în noi relații. Deoarece grupurile repetitive reprezintă un set de atribute care se repetă *împreună*, toate atributele dintr-un grup repetitiv ar trebui mutate în aceeași relație. Pe de altă parte, un atribut multivaloare (un atribut

individual care are valori multiple) ar trebui să fie mutat într-o nouă relație proprie, nu să fie combinat cu alte atribute multivaloare în noua relație.

Procedura de mutare a unui atribut multivaloare sau a unui grup repetitiv într-o nouă relație constă în următoarele etape:

1. Creați o nouă relație, cu un nume sugestiv. Deseori, este bine să includeți numele relației originale, parțial sau în întregime, în numele noii relații.
2. Copiați identificatorul unic din prima relație în noua relație. Datele depind de acest identificator în relația originală, așa că trebuie să depindă de aceeași cheie și în noua relație. Identificatorul copiat va deveni cheie externă în noua relație.
3. Mutați grupul repetitiv sau atributul multivaloare în noua relație. (Am folosit aici verbul *a muta*, deoarece aceste atribute sunt *șterse* din relația originală.)
4. Formați un identificator unic în noua relație, adăugând atribute la identificatorul unic copiat din relația originală. Ca întotdeauna, asigurați-vă că identificatorul unic nou format conține numai numărul minim de atribute necesar pentru a-l face unic. Dacă mutați un atribut multivaloare, care, în esență, este un grup repetitiv cu un singur atribut, este adăugat atributul respectiv pentru formarea identificatorului unic. Poate părea ciudat la prima vedere, dar identificatorul unic copiat din relația originală nu este doar o cheie externă, ci, de obicei, și o parte a identificatorului unic (cheia primară) a noii relații. Acest lucru este absolut normal. De asemenea, este perfect acceptabil să avem o relație în care toate atributele fac parte din identificatorul unic (adică nu există atribute care să nu facă parte din cheie).
5. Opțional, puteți să înlocuiți cheia primară cu un singur atribut surogat pentru cheie. Dacă faceți acest lucru, trebuie să păstrați și atributele care compun cheia primară naturală, formată la pașii 2 și 4.

Figura 1-5 prezintă rezultatul aducerii relației din figura 1-4 la prima formă normală. Observați următoarele:

- Am folosit o mică scurtătură în cazul identificatorului unic in noua relație Movie Language. Limba în care este disponibil un film se aplică filmului, în general, nu copiilor individuale. Observați în figura 1-4 că lista de limbi disponibile nu se schimbă între rândurile duplicate ale aceluiași film. Ca urmare, partea Copy Number a identificatorului unic din relația Movie *nu* a fost copiată în noua relație Movie Language. Dacă aș fi făcut acest lucru, aș fi creat în noua relație o problemă specifică celei de-a doua forme normale, pe care ar fi trebuit să o rezolv în următoarea etapă a procesului de normalizare. Veți descoperi că, deseori, proiectanții experimentați de baze de date sintetizează cele trei forme normale și rescriu relațiile originale direct în a treia formă normală. Exersând, veți putea și dumneavoastră să faceți același lucru.

Movie:

MOVIE ID	COPY NUMBER	GENRE CODE	GENRE DESC.	MPAA RATING CODE	MPAA RATING DESC.	MOVIE TITLE	YEAR PRODUCED	DATE ACQUIRED	DATE SOLD	MEDIA FORMAT	RETAIL PRICE
1	1	Drama	Drama	R	Under 17 requires accompanying parent or adult guardian	Mystic River	2003	1/1/2005		DVD	19.96
2	1	ActAd	Action and Adventure	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		DVD	19.96
2	2	ActAd	Action and Adventure	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	1/10/2005		VHS	15.95
3	1	Comdy	Comedy	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	1/10/2005	1/30/2005	DVD	29.99
3	2	Comdy	Comedy	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	2/15/2005		DVD	29.99
4	1	ActAd	Action and Adventure	PG-13	Parents strongly cautioned	The Italian Job	2003	2/15/2005		DVD	19.99

Movie Language:

MOVIE ID	LANGUAGE CODE
1	en
1	fr
2	en
2	fr
2	es
3	en
4	en
4	fr

Figura 1-5 Soluția primei forme normale

- Atributul Movie ID a fost copiat din relația originală (Movie) în noua relație (Movie Language).
- Atributul multivaloare Lang. Code a fost mutat din relația Movie în relația Movie Language, cu numele Language Code. (Numele abreviate ale atributelor din figura 1-4 au fost folosite doar pentru ilustrare – este recomandabil să prescurtați numele numai dacă este absolut necesar.)
- Identificatorul unic din relația Movie Language este format prin combinarea atributelor Movie ID și Language Code, adică din toate atributele relației.
- Nici Movie, nici Movie Language din figura 1-5 nu conțin grupuri repetitive sau atribute multivaloare, așa că ambele relații sunt în prima formă normală.

A doua formă normală: eliminarea dependențelor parțiale

Înainte de a explora a doua formă normală, trebuie să înțelegeți conceptul de *dependență funcțională*. Pentru această definiție, voi folosi două atribute arbitrare, inteligent denumite „A" și „B". Atributul B este *dependent funcțional* de atributul A dacă în nici un moment nu există mai mult de o valoare a atributului B asociată cu o valoare dată a atributului A. Ca să nu vă întrebați pe ce planetă a trăit autorul înainte de a-și face apariția aici, să încercăm să facem această definiție mai ușor de înțeles. În primul rând, a spune că atributul B este funcțional dependent de atributul A înseamnă și că atributul A *determină* atributul B sau că A este un *determinant* (identificator unic) pentru atributul B. În al doilea rând, să ne mai uităm o dată la relațiile din figura 1-5.

În relația Movie, puteți să vă dați seama cu ușurință că atributul Movie Title este dependent funcțional de atributul Movie ID, deoarece, în orice moment, poate exista o singură valoare Movie Title pentru o valoare Movie ID dată. Chiar faptul că valoarea Movie ID definește în mod unic valoarea Movie Title în relație înseamnă că Movie Title este dependent funcțional de Movie ID.

Se spune că o relație este în *a doua formă normală* dacă îndeplinește următoarele criterii:

- Relația este în prima formă normală.
- Toate atributele non-cheie sunt dependente funcțional de identificatorul unic (cheia primară), *luat ca întreg*.

Aplicând aceste criterii relației Movie din figura 1-5, este clar că avem câteva probleme. Identificatorul unic este o combinație a atributelor Movie ID și Copy Number. Totuși, numai atributele Date Acquired, Date Sold, Media Format și Retail Price depind de întregul identificator. Și este logic să fie așa. Indiferent câte copii ale unui film avem în baza de date, toate au aceleași valori pentru gen, categorie MPAA, titlu și an de producție. Cum a apărut această problemă? Ar trebui să fie clar că unele atribute descriu filmul în sine, în timp ce altele descriu copiile pe care le deține (sau le-a deținut) magazinul din filmul respectiv. În esență, am amestecat atribute care descriu în aceeași relație două lucruri (entități) diferite (deși înrudite) din lumea reală. Nici nu e de mirare că am obținut un asemenea haos. A doua formă normală ne va ajuta să rezolvăm problemele.

Probabil v-ați dat seama deja că a doua formă normală se aplică numai relațiilor care au identificatoare unice concatenate (adică formate din atribute multiple). Într-o relație care are un singur atribut ca identificator unic, este imposibil ca un alt atribut să depindă de o parte a identificatorului unic, deoarece acesta, fiind format dintr-un singur atribut, nu are părți componente. Ca urmare, orice relație în prima formă normală care are cheia primară formată dintr-un singur atribut este *automat* în a doua formă normală.

După ce descoperiți o încălcare a celei de-a doua forme normale, soluția este să mutați atributele parțial dependente într-o nouă relație, în care să depindă de *întreaga* cheie primară. Figura 1-6 prezintă această soluție. Toate atributele care depind numai de Movie ID sunt acum într-o relație (numită Movie) în care Movie ID este identificator unic. Cele care

depind de combinația Movie ID și Copy Number sunt într-o relație (numită Movie Copy) în care Movie ID și Copy Number formează identificatorul unic. Relația Movie Language era deja în a doua formă normală, deoarece nu are atribute non-cheie și, ca urmare, a rămas nemodificată.

A treia formă normală: eliminarea dependențelor tranzitive

Pentru a înțelege a treia formă normală, trebuie să înțelegeți mai întâi conceptul de dependență tranzitivă. Despre un atribut care depinde de un atribut care nu este identificator unic (cheie primară) a relației se spune că este *dependent tranzitiv*. Uitându-ne la relația Movie din figura 1-6, observăm că atributul Genre Description depinde de atributul Genre Code, iar MPAA Rating Description depinde de MPAA Rating Code. Pericolul păstrării acestor descrieri în relația Movie este faptul că, în final, cele două atribute ajung să depindă de înregistrarea unui film, ceea ce duce la toate cele trei anomalii de date prezentate mai devreme în acest capitol.

Se spune că o relație este în *a treia formă normală* dacă îndeplinește următoarele două criterii:

- Relația este în a doua formă normală.
- Nu există dependențe tranzitive (cu alte cuvinte, toate atributele non-cheie depind *numai* de identificatorul unic).

Pentru a aduce la a treia formă normală o relație aflată în a doua formă normală, mutați atributele dependente tranzitiv în relații în care depind numai de cheia primară. Aveți grijă să lăsați atributul de care depind acestea în relația originală, cu rolul de cheie externă. Va trebui apoi să reconstruiți vizualizarea originală printr-o uniune. Ca efect secundar, toate atributele ușor de calculat sunt eliminate ca încălcări ale criteriilor celei de-a treia forme normale. De exemplu, într-o bază de date pentru vânzări, Suma Totală este obținută înmulțind Cantitatea Cumpărată cu Prețul Unitar; așa cum se observă cu ușurință, Suma Totală este dependentă de Cantitatea Cumpărată și de Prețul Unitar. Presupunând că toate cele trei atribute sunt dependente de identificatorul unic al relației care le conține, este ușor de văzut că Suma Totală (rezultatul calculat) este, de fapt, *dependentă tranzitiv* de celelalte două atribute.

Figura 1-7 conține soluția în a treia formă normală. Remarcați că am creat noi relații pentru MPAA Rating și Movie Genre, am mutat descrierile în noile relații și am lăsat atributele pentru coduri (MPAA Rating Code și Movie Genre Code) în relația Movie, definite ca fiind chei externe. Mulți proiectanți de baze de date numesc relațiile MPAA Rating și Movie Genre „tabele de căutare" sau „tabele de coduri", deoarece sunt utilizate, în principal, pentru căutarea codurilor stocate în coloana cheie primară a relației. Totuși, aceste relații au și alte roluri, cum ar fi controlul codurilor și furnizarea unei surse convenabile pentru lista de coduri valide, care poate fi folosită într-o listă derulantă de pe o pagină web.

Movie:

MOVIE ID	GENRE CODE	GENRE DESCRIPTION	MPAA RATING CODE	MPAA RATING DESCRIPTION	MOVIE TITLE	YEAR PRODUCED
1	Drama	Drama	R	Under 17 requires accompanying parent or adult guardian	Mystic River	2003
2	ActAd	Action and Adventure	R	Under 17 requires accompanying parent or adult guardian	The Last Samurai	2003
3	Comdy	Comedy	PG-13	Parents strongly cautioned	Something's Gotta Give	2003
4	ActAd	Action and Adventure	PG-13	Parents strongly cautioned	The Italian Job	2003

Movie Language:

MOVIE ID	LANGUAGE CODE
1	en
1	fr
2	en
2	fr
2	es
3	en
4	en
4	fr

Movie Copy:

MOVIE ID	COPY NUMBER	DATE ACQUIRED	DATE SOLD	MEDIA FORMAT	RETAIL PRICE
1	1	1/1/2005		DVD	19.96
2	1	1/10/2005		DVD	19.96
2	2	1/10/2005		VHS	15.95
3	1	1/10/2005	1/30/2005	DVD	29.99
3	2	2/15/2005		DVD	29.99
4	1	2/15/2005		DVD	19.99

Figura 1-6 Soluția pentru a doua formă normală

Movie:

MOVIE ID	MOVIE GENRE CODE	MPAA RATING CODE	MOVIE TITLE	RETAIL PRICE VHS	RETAIL PRICE DVD	YEAR PRODUCED
1	Drama	R	Mystic River	58.97	19.96	2003
2	ActAd	R	The Last Samurai	15.95	19.96	2003
3	Comdy	PG-13	Something's Gotta Give	14.95	29.99	2003
4	ActAd	PG-13	The Italian Job	11.95	19.99	2003

Movie Copy:

MOVIE ID	COPY NUMBER	DATE ACQUIRED	DATE SOLD	MEDIA FORMAT
1	1	1/1/2005		DVD
2	1	1/10/2005		DVD
2	2	1/10/2005		VHS
3	1	1/10/2005	1/30/2005	DVD
3	2	2/15/2005		DVD
4	1	2/15/2005		DVD

MPAA Rating:

MPAA RATING CODE	MPAA RATING DESCRIPTION
PG-13	Parents strongly cautioned
R	Under 17 requires accompanying parent or adult guardian

Movie Genre:

MOVIE GENRE CODE	MOVIE GENRE DESCRIPTION
ActAd	Action and Adventure
Comdy	Comedy
Drama	Drama

Figura 1-7 Soluția pentru a treia formă normală

O altă modificare făcută pentru a ajunge la a treia formă normală este legată de atributul Retail Price (preț cu amănuntul) din relația Movie Copy, așa cum se poate vedea în figura 1-6. După o discuție cu proprietarul magazinului, am stabilit că prețul depinde de combinația dintre Movie ID și Media Format, toate copiile cu aceleași valori pentru Movie ID și Media Format având același preț. În mod clar, aceasta este o dependență tranzitivă și, ca urmare, o încălcare a celei de-a treia forme normale. Soluția normală pentru o asemenea problemă ar fi crearea unei relații numite Movie Price, având ca identificator unic combinația dintre Movie ID și Media Format, și mutarea atributului Retail Price din Movie Copy în noua relație. Totuși, în timpul discuției cu managerul magazinului am aflat că urmează să renunțe la furnizarea filmelor în format VHS, deoarece sunt cerute de un număr foarte mic de clienți și că peste câteva luni magazinul va avea numai filme pe DVD. Ținând seama de această informație, am decis să mut prețul în două coloane din tabelul Movie, una cu prețul pentru DVD și una cu prețul pentru VHS. Deși se poate spune că aceasta este o încălcare a primei forme normale (și, din punct de vedere tehnic chiar este), mi s-a părut a fi cel mai bun compromis. Proiectarea bazelor de date nu este întotdeauna o știință exactă, așa că de multe ori există posibilitatea unor mici ajustări, cu condiția ca proiectantul să ia în calcul consecințele potențiale (măsurate în termenii anomaliilor de date) ale fiecărui compromis.

Dincolo de a treia formă normală

Dr. E. F. Codd a participat la definirea unei versiuni mai puternice a celei de-a treia forme normale, numită forma normală Boyce-Codd. Diferiți alți autori și cercetători au oferit propriile extensii, numite a patra formă normală, a cincea formă normală, forma normală cu cheie de domeniu și altele. Din experiența mea, este nevoie de ceva exercițiu în procesul de normalizare înainte ca aceste extensii să devină logice. În plus, a treia formă normală acoperă aproape toate anomaliile pe care este posibil să le întâlniți în lucrul cu baze de date obișnuite.

Prezentare generală a bazei de date pentru un magazin de produse video

Toate exemplele din această carte folosesc ca exemplu o bază de date pentru un magazin fictiv de produse video. Instrucțiunile SQL necesare pentru crearea obiectelor bazei de date și pentru popularea acestora cu date pot fi descărcate conform explicațiilor din acest capitol. Figura 1-8 prezintă diagrama de relații a entităților (ERD – Entity Relationship Diagram) pentru întreaga bază de date a magazinului de produse video. În continuare veți găsi o scurtă descriere a fiecărui tabel, astfel încât să vă familiarizați cu modelul bazei de date, ceea ce vă va ajuta să înțelegeți exemplele din următorul capitol, când veți începe să învățați limbajul SQL.

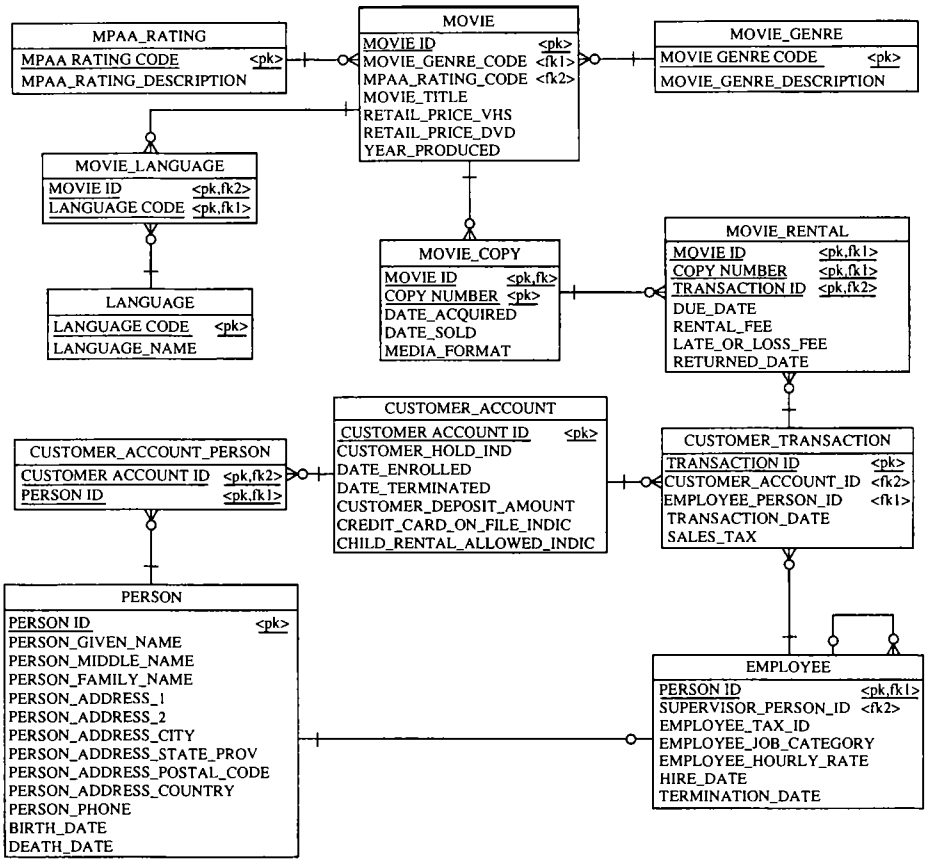


Figura 1-8 Diagrama ERD a magazinului de produse video

Magazinul pentru care a fost proiectată această bază de date este un mic magazin, deținut și administrat de o familie, și se ocupă de închirierea și comercializarea produselor video în format VHS și DVD. Totuși, magazinul este pe cale de a renunța la desfacerea filmelor în format VHS. Proprietarul magazinului intenționează să adauge și alte categorii de produse, cum ar fi cele alimentare, dar aceste intenții nu sunt încă suficient de clare pentru a fi incluse în proiectul curent al bazei de date.

Tabelul 1-1 vă oferă câteva informații despre fiecare tabel din proiectul bazei de date.

Tabelul 1-2 vă prezintă pe scurt coloanele din fiecare tabel al bazei de date.

Numele tabelului	Descriere	Cheie primară	Tabelele părinte
CUSTOMER_ ACCOUNT	Conține un rând pentru fiecare cont de client deschis în magazinul de produse video.	CUSTOMER_ ACCOUNT_ID	Nu există
CUSTOMER_ ACCOUNT_ PERSON	Tabel de intersectare care arată ce persoane sunt asociate cu fiecare cont de client.	CUSTOMER_ ACCOUNT_ID, PERSON_ID	CUSTOMER_ ACCOUNT, PERSON
CUSTOMER_ TRANSACTION	Conține un rând pentru fiecare tranzacție inițiată de un client. Fiecare tranzacție poate conține una sau mai multe închirieri de filme.	TRANSACTION_ID	CUSTOMER_ ACCOUNT, EMPLOYEE
EMPLOYEE	Conține un rând pentru fiecare angajat al magazinului de produse video. Acest tabel este o subclasă a tabelului PERSON (fiecare angajat va avea un rând corespondent, cu aceeași cheie primară în tabelul PERSON).	PERSON_ID	PERSON
LANGUAGE	Tabel de căutare pentru codurile și numele limbilor (folosit pentru a afișa opțiunile de limbă ale filmelor).	LANGUAGE_CODE	Nu există.
MOVIE	Conține un rând pentru fiecare titlu de film. Tabelul „copil" MOVIE_COPY conține copiile filmelor aflate în magazin.	MOVIE_ID	MPAA_RATING, MOVIE_GENRE
MOVIE_COPY	Conține un rând pentru fiecare copie a unui film disponibilă pentru vânzare sau închiriere.	MOVIE_ID, COPY_NUMBER	MOVIE
MOVIE_GENRE	Tabel de căutare pentru coduri și descrieri (folosit pentru a împărți filmele pe genuri).	MOVIE_GENRE_ CODE	Nu există.
MOVIE_ LANGUAGE	Tabel de intersectare care prezintă limbile disponibile pentru fiecare film.	MOVIE_ID, LANGUAGE_ CODE	MOVIE, LANGUAGE

Tabelul 1-1 Tabelele din baza de date a magazinului de produse video

Numele tabelului	Descriere	Cheie primară	Tabel(e) părinte
MOVIE_RENTAL	Conține un rând pentru fiecare închiriere a unui film.	MOVIE_ID, COPY_NUMBER, TRANSACTION_ID	MOVIE_COPY, CUSTOMER_ TRANSACTION
MPAA_RATING	Tabel de căutare pentru codurile și descrierile MPAA.	MPAA_RATING_ CODE	Nu există.
PERSON	Conține un rând pentru fiecare persoană asociată cu magazinul de produse video.	PERSON_ID	Nu există.

Tabelul 1-1 Tabelele din baza de date a magazinului de produse video *(continuare)*

Numele tabelului	Numele coloanei	Descriere
CUSTOMER_ACCOUNT	CHILD_RENTAL_ ALLOWED_INDIC	Indicator de tip Da/Nu care arată dacă persoanele sub 18 ani au voie să închirieze filme folosind acest cont.
CUSTOMER_ACCOUNT	CREDIT_CARD_ON_FILE_ INDIC	Indicator de tip Da/Nu care arată dacă clientul respectiv a lăsat un dosar cu informații despre cartea de credit pentru a garanta plățile asociate contului.
CUSTOMER_ACCOUNT	CUSTOMER_ACCOUNT_ID	Cheie primară – un număr secvențial asociat fiecărui cont de client.
CUSTOMER_ACCOUNT	CUSTOMER_DEPOSIT_ AMOUNT	Pentru clienții care nu au furnizat o carte de credit, suma lăsată ca depozit în magazin.
CUSTOMER_ACCOUNT	CUSTOMER_HOLD_IND	Indicator de tip Da/Nu care arată dacă un cont este blocat; închirierile nu sunt permise pentru conturile blocate.
CUSTOMER_ACCOUNT	DATE_ENROLLED	Data la care a fost deschis contul în magazin.
CUSTOMER_ACCOUNT	DATE_TERMINATED	Dacă un cont a fost închis, conține data închiderii contului (conține valoarea null pentru conturile active).

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video

Numele tabelului	Numele coloanei	Descriere
CUSTOMER_ACCOUNT_ PERSON	CUSTOMER_ACCOUNT_ID	Parte a cheii primare – cheie externă către tabelul CUSTOMER_ ACCOUNT.
CUSTOMER_TRANSACTION	EMPLOYEE_PERSON_ID	Cheie externă (PERSON_ID) către tabelul EMPLOYEE.
CUSTOMER_TRANSACTION	SALES_TAX	Taxa de vânzare încasată pentru tranzacție.
CUSTOMER_TRANSACTION	TRANSACTION_DATE	Data tranzacției.
CUSTOMER_TRANSACTION	TRANSACTION_ID	Conține un rând pentru fiecare tranzacție inițiată de un client. Fiecare tranzacție poate conține una sau mai multe închirieri de filme.
CUSTOMER_TRANSACTION	TRANSACTION_ID	Cheie primară – un număr secvențial asociat fiecărei noi tranzacții.
EMPLOYEE	EMPLOYEE_HOURLY_RATE	Plata pe oră pentru fiecare angajat.
EMPLOYEE	EMPLOYEE_JOB_CATEGORY	Categoria de încadrare a fiecărui angajat (manager sau funcționar).
EMPLOYEE	EMPLOYEE_TAX_ID	Numărul de identificare (ID) pentru raportarea taxelor plătite pentru fiecare angajat (de obicei, un număr de asigurări sociale).
EMPLOYEE	HIRE_DATE	Data la care a fost angajat în magazin.
EMPLOYEE	PERSON_ID	Cheie primară – cheie externă către tabelul PERSON.
EMPLOYEE	SUPERVISOR_PERSON_ID	Cheie primară – cheie externă către tabelul EMPLOYEE (pentru a identifica persoana căreia i se subordonează).

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video *(continuare)*

Numele tabelului	Numele coloanei	Descriere
EMPLOYEE	TERMINATION_DATE	Pentru foștii angajați, data la care au încetat să lucreze pentru magazin.
LANGUAGE	LANGUAGE_CODE	Cheie primară – codul standard ISO (International Organization for Standardization) de două caractere pentru limbă.
LANGUAGE	LANGUAGE_NAME	Numele limbii (în engleză).
MOVIE	MOVIE_GENRE_CODE	Cheie externă către tabelul MOVIE_GENRE.
MOVIE	MOVIE_ID	Cheie primară – valorile sunt atribuite secvențial, pe măsură ce noile filme devin disponibile.
MOVIE	MOVIE_TITLE	Titlul oficial al filmului (nu este obligatoriu ca titlurile filmelor să fie unice).
MOVIE	MOVIE_RATING_CODE	Cheie externă către tabelul MPAA_RATING.
MOVIE	RETAIL_PRICE_DVD	Prețul de vânzare cu amănuntul pentru copiile în format DVD ale filmului.
MOVIE	RETAIL_PRICE_VHS	Prețul de vânzare cu amănuntul pentru copiile în format VHS ale filmului.
MOVIE	YEAR_PRODUCED	Anul în care s-a terminat producția filmului; anul specificat de studio.
MOVIE_COPY	COPY_NUMBER	Parte a cheii primare – un număr secvențial asociat fiecărei copii a unui film (numerele sunt unice numai pentru un anumit film).
MOVIE_COPY	DATE_ACQUIRED	Data la care a fost obținută de magazin copia respectivă a filmului.

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video *(continuare)*

Numele tabelului	Numele coloanei	Descriere
MOVIE_COPY	DATE_SOLD	Data la care a fost vândută copia filmului (are valoarea null dacă filmul nu a fost vândut).
MOVIE_COPY	MEDIA_FORMAT	Formatul de înregistrare a copiei filmului (DVD sau VHS).
MOVIE_COPY	MOVIE_ID	Parte a cheii primare – cheie externă către tabelul MOVIE.
MOVIE_GENRE	MOVIE_GENRE_CODE	Cheie primară – un cod folosit pentru a plasa filmele în diferite categorii, cum ar fi comedie, dramă, acțiune-aventură și așa mai departe.
MOVIE_GENRE	MOVIE_GENRE_DESCRIPTION	Descriere sub formă de text a categoriei filmului (a se vedea coloana MOVIE_GENRE_ CODE).
MOVIE_LANGUAGE	LANGUAGE_CODE	Parte a cheii primare – cheie externă către tabelul LANGUAGE.
MOVIE_LANGUAGE	MOVIE_ID	Parte a cheii primare – cheie externă către tabelul MOVIE.
MOVIE_RENTAL	COPY_NUMBER	Parte a cheii primare – cheie externă către tabelul MOVIE_COPY.
MOVIE_RENTAL	DUE_DATE	Data la care filmul închiriat trebuie să fie returnat la magazin.
MOVIE_RENTAL	LATE_OR_LOSS_FEE	Suprataxa (dacă există) percepută pentru returnarea cu întârziere sau pierderea filmului.
MOVIE_RENTAL	MOVIE_ID	Parte a cheii primare – cheie externă către tabelul MOVIE_COPY.

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video *(continuare)*

Numele tabelului	Numele coloanei	Descriere
MOVIE_RENTAL	RENTAL_FEE	Taxa de închiriere (ajustată cu eventualele cupoane sau reduceri).
MOVIE_RENTAL	RETURNED_DATE	Data la care a fost returnată copia filmului (conține valoarea null până la returnarea filmului).
MOVIE_RENTAL	TRANSACTION_ID	Parte a cheii primare – cheie externă către tabelul CUSTOMER_ TRANSACTION.
MPAA_RATING	MPAA_RATING_CODE	Cheie primară – codurile categoriilor de încadrare ale filmelor, furnizate de MPAA (Motion Picture Association of America), inclusiv G, PG, PG-13, R, NC-17 și NR.
MPAA_RATING	MPAA_RATING_DESCRIPTION	Descrierea sub formă de text a fiecărei categorii MPAA.
PERSON	BIRTH_DATE	Data de naștere a persoanei.
PERSON	DEATH_DATE	Data decesului (dacă persoana este raportată ca decedată).
PERSON	PERSON_ADDRESS_1	Prima linie pentru adresa persoanei.
PERSON	PERSON_ADDRESS_2	A doua linie, opțională, pentru adresa persoanei.
PERSON	PERSON_ADDRESS_CITY	Orașul din adresa poștală a persoanei.
PERSON	PERSON_ADDRESS_COUNTRY	Abrevierea ISO pentru țara din adresa poștală a persoanei.
PERSON	PERSON_ADDRESS_POSTAL_ CODE	Codul poștal (codul ZIP în SUA) din adresa poștală a persoanei.
PERSON	PERSON_ADDRESS_STATE_ PROV	Statul sau provincia din adresa poștală a persoanei.

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video *(continuare)*

Numele tabelului	Numele coloanei	Descriere
PERSON	PERSON_FAMILY_NAME	Numele de familie al persoanei.
PERSON	PERSON_GIVEN_NAME	Prenumele persoanei.
PERSON	PERSON_ID	Cheie primară – un număr secvențial atribuit fiecărei persoane care are legătură cu magazinul de produse video.
PERSON	PERSON_MIDDLE_NAME	Al doilea prenume (sau inițiala) persoanei.
PERSON	PERSON_PHONE	Principalul număr de telefon al persoanei.

Tabelul 1-2 Coloanele tabelelor din baza de date a magazinului de produse video *(continuare)*

Descărcarea codului SQL pentru baza de date folosită ca exemplu

Un script care conține instrucțiunile SQL CREATE standard pentru crearea tuturor tabelelor și indexurilor din baza de date a magazinului de produse video, folosită în toate exemplele din această carte, împreună cu instrucțiunile INSERT pentru popularea tabelelor cu datele folosite în exemple, poate fi descărcat de pe site-ul web www.osborne.com*. Vă recomand să creați baza de date și să rulați exemplele pe măsură ce citiți cartea. Parcurgeți următorii pași:

1. Folosiți browser-ul web pentru a naviga la adresa www.osborne.com.
2. Executați clic pe legătura „free code" din apropierea colțului din stânga al paginii.
3. Derulați lista până la legătura „SQL Demystified" și executați clic pe aceasta.
4. Descărcarea ar trebui să înceapă automat – browser-ul vă va întreba unde doriți să salvați fișierul.

Într-o lume ideală, codul SQL care respectă standardul ISO/ANSI ar trebui să poată fi rulat cu orice bază de date care acceptă limbajul SQL. Deși nu se întâmplă astfel, vestea bună este că nu trebuie să faceți decât, cel mult, câteva modificări pentru sistemele DBMS cele mai frecvent folosite. Pentru unele sisteme DBMS, precum MySQL, nu este necesară

* La terminarea traducerii, fișierul respectiv putea fi descărcat de la adresa http://books.mcgraw-hill.com/downloads/products/0072262249/0072262249_appendix.pdf (nota trad.)

nici o modificare. Pentru Oracle și Microsoft SQL Server, modificările necesare sunt prezentate în secțiunile următoare. Pentru alte sisteme DBMS, consultați documentația producătorilor în privința tipurilor de date acceptare, în special cele referitoare la dată și oră.

Modificări necesare pentru Oracle

Formatul prestabilit pentru date calendaristice în Oracle este DD-MON-RR (ZZ-LLL-AA), adică ziua pe două cifre, luna pe trei litere și anul pe două cifre, cu secolul bazat pe o fereastră de 50 de ani. De exemplu, data de 15 ianuarie 2005 este afișată sub forma 15-JAN-05 în formatul Oracle prestabilit. Instrucțiunile SQL INSERT din această anexă și toate exemplele din această care folosesc formatul MM/DD/YYYY (LL/ZZ/AAAA), ceea ce înseamnă că data de 15 ianuarie 2005 este scrisă sub forma 01/15/2005. Există două modalități de ocolire a acestei probleme:

- **Schimbați formatul datelor** Deși acest lucru este posibil, nu este recomandabil să modificați permanent formatul prestabilit al datelor calendaristice în baza de date. Totuși, este foarte simplu să schimbați formatul numai pentru sesiunea dumneavoastră, imediat după ce vă conectați la baza de date Oracle. Tot ce trebuie să faceți este să rulați următoarea comandă în clientul Oracle SQL (SQL*Plus sau iSQL*Plus) înainte de a rula orice alte instrucțiuni care includ date:

  ```
  ALTER SESSION SET NLS_DATE_FORMAT='MM/DD/YYYY';
  ```

- **Modificați toate șirurile de date** Aceasta este o soluție mai greoaie, deoarece implică editarea tuturor instrucțiunilor SQL și modificarea tuturor datelor din formatul MM/DD/YYYY în formatul DD-MON-RR.

Modificări necesare pentru Microsoft SQL Server

Pentru Microsoft SQL Server sunt necesare următoarele modificări:

- **Înlocuiți DATE cu DATETIME** Microsoft SQL Server nu recunoaște DATE ca tip de date. Caracteristicile DBMS se schimbă permanent, așa că este posibil ca acest tip de date să fie acceptat în SQL Server 2005 sau într-o versiune ulterioară. Cel puțin până la SQL Server 2000, toate aparițiile tipului DATE din instrucțiunile CREATE TABLE trebuie să fie înlocuite cu DATETIME.
- **Pentru iSQL, puneți instrucțiunile în fișiere de comenzi (batches)** Clientul iSQL se așteaptă ca instrucțiunile SQL să fie rulate din fișiere de comenzi, cu cuvântul cheie go scris pe o linie separată la sfârșitul fiecărui fișier. În timp ce majoritatea clienților SQL rulează instrucțiunile imediat ce ajung la caracterul punct și virgulă care marchează sfârșitul instrucțiunii, iSQL așteaptă să ajungă la cuvântul cheie go. Puteți include orice număr de instrucțiuni în fișierul

de comenzi, dar un număr prea mare de instrucțiuni face ca rezolvarea problemelor să fie mai dificilă. Adăugați cuvântul cheie go periodic între instrucțiunile SQL, pe linii separate – și nu uitați să-l adăugați și la sfârșitul fișierului.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. SQL este
   a. Un limbaj inventat de Dr. E. F. Codd
   b. Un limbaj folosit pentru comunicarea cu bazele de date relaționale
   c. Un limbaj folosit pentru definirea diagramelor ERD
   d. Folosit pentru definirea și modificarea obiectelor unei baze de date
   e. Folosit pentru definirea paginilor web
2. O bază de date este
   a. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index
   b. Un produs software furnizat de un producător de baze de date
   c. O colecție de date interrelaționate gestionate ca o singură unitate
   d. Definită în același mod de toți producătorii de software
   e. Implementată diferit de diferiți producători
3. Un obiect al unei baze de date este
   a. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index
   b. O structură, cum ar fi un tabel, o vizualizare sau un index
   c. Un produs software furnizat de un producător de baze de date
   d. O colecție de date interrelaționate gestionate ca o singură unitate
   e. O colecție de înregistrări înrudite, stocate ca o singură unitate
4. Un sistem de gestionare a bazelor de date este
   a. O structură, cum ar fi un tabel, o vizualizare sau un index
   b. O colecție de date interrelaționate gestionate ca o singură unitate

c. Un produs software furnizat de un producător de baze de date

d. Deseori abreviat sub forma DBMS

e. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index

5. Exemple de sisteme RDBMS sunt

a. INGRES

b. MySQL

c. PostgreSQL

d. Oracle Database

e. Microsoft SQL Server

6. Serviciile de bază oferite de un sistem DBMS includ

a. Acceptarea unui limbaj de interogare

b. Generarea diagramelor ERD (Entity Relationship Diagram)

c. Mecanisme de securitate, pentru a împiedica accesul și modificarea neautorizată a datelor

d. Mutarea datelor în și din fișiere de date, după cum este necesar

e. Stocarea datelor într-o formă tabelară

7. Componentele unei baze de date relaționale includ

a. Relații

b. Tabele

c. Vizualizări de utilizator

d. Diagrame ERD

e. Restricții

8. Într-o diagramă ERD, cardinalitatea maximă este reprezentată prin

a. Un cerc desenat pe linia care reprezintă relația

b. O linie verticală care taie linia care reprezintă relația

c. Un simbol numit „picior de cioară" desenat la capătul liniei care reprezintă relația

d. Absența unui simbol la sfârșitul liniei care reprezintă relația

e. Notația „<pk>" deasupra liniei care reprezintă relația

9. Printre tipurile de restricții care pot fi folosite în bazele de date se numără
   a. NOT NULL
   b. Relație
   c. Cheie primară
   d. CHECK
   e. Unicitate
10. Printre funcțiile vizualizărilor se numără
   a. Documentarea relațiilor dintre tabele
   b. Creșterea performanțelor interogărilor
   c. Ascunderea coloanelor pe care nu este nevoie să le vadă utilizatorul
   d. Ascunderea rândurilor pe care nu este nevoie să le vadă utilizatorul
   e. Expunerea operațiilor complexe executate în baza de date
11. Scopul normalizării este de a rezolva următoarele probleme
   a. Anomalia de inserare
   b. Performanțe reduse
   c. Anomalia de creare
   d. Anomalia de ștergere
   e. Anomalia de actualizare
12. Procesul de normalizare
   a. Începe de la tabele, pentru a ajuta proiectanții să descopere vizualizările de utilizator
   b. A fost dezvoltat de Dr. E. F. Codd
   c. A fost dezvoltat de Oracle
   d. Este aplicat sistematic fiecărei vizualizări de utilizator
   e. Este ușor de stăpânit
13. Un identificator unic
   a. Trebuie să fie găsit înainte de a se putea face normalizarea unei vizualizări
   b. Poate fi compus dintr-un singur atribut

c. Poate fi format prin inventarea unui nou atribut
d. Poate fi compus din zero, unul sau mai multe atribute
e. Poate fi compus din atribute concatenate

14. Prima formă normală rezolvă anomaliile cauzate de
    a. Dependențele parțiale de cheia primară
    b. Grupurile repetitive
    c. Dependențele tranzitive
    d. Atributele multivaloare
    e. Relațiile de tip unu-la-mai-mulți
15. A doua formă normală rezolvă anomaliile cauzate de
    a. Dependențele parțiale de cheia primară
    b. Grupurile repetitive
    c. Dependențele tranzitive
    d. Atributele multivaloare
    e. Relațiile de tip unu-la-mai-mulți
16. A treia formă normală rezolvă anomaliile cauzate de
    a. Dependențele parțiale de cheia primară
    b. Grupurile repetitive
    c. Dependențele tranzitive
    d. Atributele multivaloare
    e. Relațiile de tip unu-la-mai-mulți
17. Pentru a fi în a treia formă normală, o relație
    a. Trebuie să fie în prima formă normală
    b. Trebuie să fie în a doua formă normală
    c. Trebuie să aibă un identificator unic
    d. Trebuie să nu aibă dependențe tranzitive
    e. Trebuie să nu aibă grupuri repetitive sau atribute multivaloare

18. Pentru aducerea în prima formă normală a unei relații ne-normalizate
    a. Atributele care depind doar de o parte a cheii sunt eliminate
    b. Atributele multivaloare sunt mutate într-o nouă relație
    c. Atributele care sunt dependente tranzitiv sunt eliminate
    d. Grupurile repetitive sunt mutate într-o nouă relație
    e. Identificatorul unic al relației originale este copiat în noua relație
19. Problemele de dependență parțială
    a. Sunt rezolvate de a doua formă normală
    b. Sunt rezolvate de a treia formă normală
    c. Pot apărea numai în relațiile cu chei primare concatenate
    d. Apar atunci când un atribut non-cheie depinde doar de o parte a cheii primare
    e. Apar atunci când un atribut non-cheie depinde de un alt atribut non-cheie
20. Problemele de dependență tranzitivă
    a. Sunt rezolvate de a doua formă normală
    b. Sunt rezolvate de a treia formă normală
    c. Pot apărea numai în relațiile cu chei primare concatenate
    d. Apar atunci când un atribut non-cheie depinde doar de o parte a cheii primare
    e. Apar atunci când un atribut non-cheie depinde de un alt atribut non-cheie

CAPITOLUL 2

Concepte SQL

Acest capitol vă face cunoștință cu limbajul SQL, într-o discuție pe care o vom extinde apoi pe parcursul întregi cărți. SQL a devenit limbajul universal pentru bazele de date relaționale și este acceptat de aproape toate sistemele RDBMS moderne. Fără îndoială, acceptarea pe scară largă este rezultatul timpului și eforturilor depuse pentru dezvoltarea caracteristicilor limbajului și a standardelor, crescând nivelul de portabilitate a codului SQL între diferitele produse RDBMS.

Veți avea mai multe succes în învățarea limbajului SQL dacă folosiți în mod activ SQL și o bază de date relațională pentru a încerca exemplele din această carte pe măsură ce învățați. Mai mult decât atât, veți învăța și mai bine dacă veți încerca variante proprii ale exemplelor. Puteți implementa baza de date a magazinului de produse video, folosită ca exemplu în toată această carte, în orice sistem de baze de date relaționale doriți. Pașii pe care trebuie să-i urmați pentru descărcarea instrucțiunilor care definesc tabelele și le încarcă cu date sunt prezentați undeva către sfârșitul capitolului 1.

Ce este SQL?

SQL (Structured Query Language – limbaj structurat de interogare) este un limbaj standard folosit pentru comunicarea cu bazele de date. Numele limbajului poate fi pronunțat pe litere (es-q-el) sau la fel ca și cuvântul englezesc „sequel". O *interogare (query)* este o simplă cerere transmisă către baza de date, la care aceasta răspunde într-o anumită formă. SQL este limbajul folosit cel mai frecvent pentru interogarea bazelor de date. SQL este considerat un limbaj *neprocedural* sau *declarativ*, ceea ce înseamnă că-i spuneți calculatorului ce rezultate vreți, fără să-i spuneți cum să le obțină. De exemplu, dacă vreți să obțineți media numerelor de pe o coloană, folosiți funcția AVG. Nu este nevoie să numărați valorile din coloană și să împărțiți suma acestora la numărul obținut – procesorul limbajului SQL din DBMS se ocupă de toate aceste lucruri în locul dumneavoastră. Funcțiile SQL sunt prezentate în detaliu în capitolul 4.

Este important să înțelegi că SQL nu este un limbaj procedural, ca C, Pascal, Basic, FORTRAN, COBOL sau Ada. Un limbaj *procedural* folosește o serie de instrucțiuni executate secvențial. De asemenea, limbajele procedurale includ instrucțiuni care pot modifica secvența de execuție, prin ramificarea la alte porțiuni ale procedurii sau prin parcurgerea ciclică a unui set de instrucțiuni din procedură. Mulți producători de sisteme RDBMS oferă extensii procedurale ale limbajului SQL de bază, cum ar fi Oracle PL/SQL (Procedural Language/SQL) sau Microsoft Transact-SQL, dar rețineți că acestea sunt extensii SQL care formează noi limbaje – codul SQL pe care-l conțin rămâne neprocedural. De asemenea, SQL nu trebuie confundat cu limbajele orientate spre obiecte, precum Java și C++. Simplu spus, SQL este un limbaj pentru gestionarea și întreținerea bazelor de date relaționale, nu un limbaj potrivit pentru programarea generală a aplicațiilor, cum ar fi sistemele de prelucrare a comenzilor sau a plăților.

SQL este deseori folosit în combinație cu limbajele procedurale sau orientate spre obiecte menționate anterior pentru a manipula stocarea și extragerea datelor, folosind instrucțiuni din limbajul de programare cu destinație generală pentru alte sarcini de programare, precum prezentarea datelor pe o pagină web sau furnizarea răspunsurilor la informațiile introduse de utilizatori de la tastatură sau mouse. Atunci când este necesară interacționarea cu baza de date, instrucțiunile din limbajul procedural formează instrucțiunea SQL, o transmit către RDBMS în vederea prelucrării, primesc rezultatele returnate de RDBMS și le prelucrează într-un mod corespunzător.

Conectarea la baza de date

Atunci când folosiți limbajul SQL pe un calculator personal, cu o copie personală a unui sistem DBMS, precum Microsoft Access sau Oracle Personal Edition, toate componentele bazei de date rulează pe același sistem de calcul. Totuși, acest aranjament nu este potrivit pentru bazele de date care trebuie să fie folosite în comun de mai mulți utilizatori. Ca

...mare, sunt mult mai frecvent întâlnite situațiile în care baza de date este instalată într-un ...anjament client/server, ca în figura 2-1.

Într-un aranjament client/server:

- Sistemul RDBMS rulează pe un *server*, care este un sistem de calcul partajat. Pentru scopurile acestei definiții, un sistem mainframe poate fi considerat un server de dimensiuni mari.
- Fișierele care compun baza de date din punct de vedere fizic sunt stocate pe discuri conectate la serverul de baze de date.
- Utilizatorii care au acces la baza de date folosesc stații de lucru, numite *clienți*. Clientul trebuie să aibă o conexiune de rețea la baza de date, care poate fi o rețea privată, instalată acasă sau la birou, ori o rețea publică, precum Internet.
- Componentele software furnizate de producătorul DBMS rulează pe stațiile de lucru alte clienților pentru a oferi utilizatorilor posibilitatea să introducă instrucțiuni SQL, să le transmită sistemului DBMS în vederea prelucrării și să vadă rezultatele returnate de DBMS. În general, acest software se numește client SQL.

Rețineți că nimic nu vă oprește să instalați clientul SQL pe același calculator cu sistemul DBMS. De fapt, mulți dezvoltatori care utilizează sisteme DBMS precum MySQL, Microsoft SQL Server și Oracle fac în mod obișnuit acest lucru, deoarece este foarte

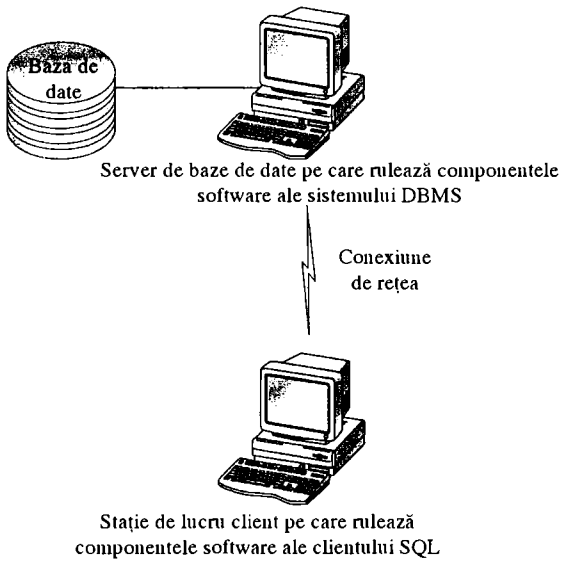


Figura 2-1 Conexiunea clientului SQL cu baza de date

convenabil să aibă întregul mediu de dezvoltare pe un singur calculator, cum ar fi un laptop. Totuși, în momentul în care este necesar accesul partajat al mai multor utilizatori, este mult mai convenabil și mai eficient să aveți o singură copie a sistemului DBMS pe un server partajat și să aveți numai clientul SQL instalat pe stația de lucru a fiecărui client.

În funcție de interfața cu utilizatorul de pe stația de lucru client, clienții SQL sunt clasificați în trei categorii: în linia de comandă, grafici și bazați pe web. O interfață *în linia de comandă* se bazează exclusiv pe intrări și ieșiri de tip text, cu comenzile introduse de la tastatură și răspunsurile afișate ca mesaje de tip text. Principalul avantaj al interfețelor în linia de comandă este că pot fi rulate pe aproape orice sistem de operare. Ca exemplu de client SQL în linia de comandă, figura 2-2 prezintă SQL*Plus (unul din clienții SQL furnizați de Oracle) rulat într-o fereastră de comandă Microsoft Windows. O *interfață grafică cu utilizatorul (GUI – graphical user interface)* rulează sub un tip oarecare de sistem bazat pe ferestre, cum ar fi X Window System, Mac OS sau Microsoft Windows, și afișează datele sau opțiunile comenzilor folosind elemente grafice, precum pictograme, butoane și casete de dialog. Figura 2-3 prezintă clientul SQL*Plus rulat ca aplicație GUI sub Microsoft Windows. O *interfață bazată pe web* rulează pe serverul de baze de date, folosind un browser web de pe stația de lucru client pentru a interacționa cu utilizatorul bazei de date. Din punct de vedere tehnic, un client SQL bazat pe web nici nu este o aplicație client, deoarece nu există nici o componentă software specifică producătorului DBMS rulată pe stația de lucru a clientului. Totuși, aproape întotdeauna există componente

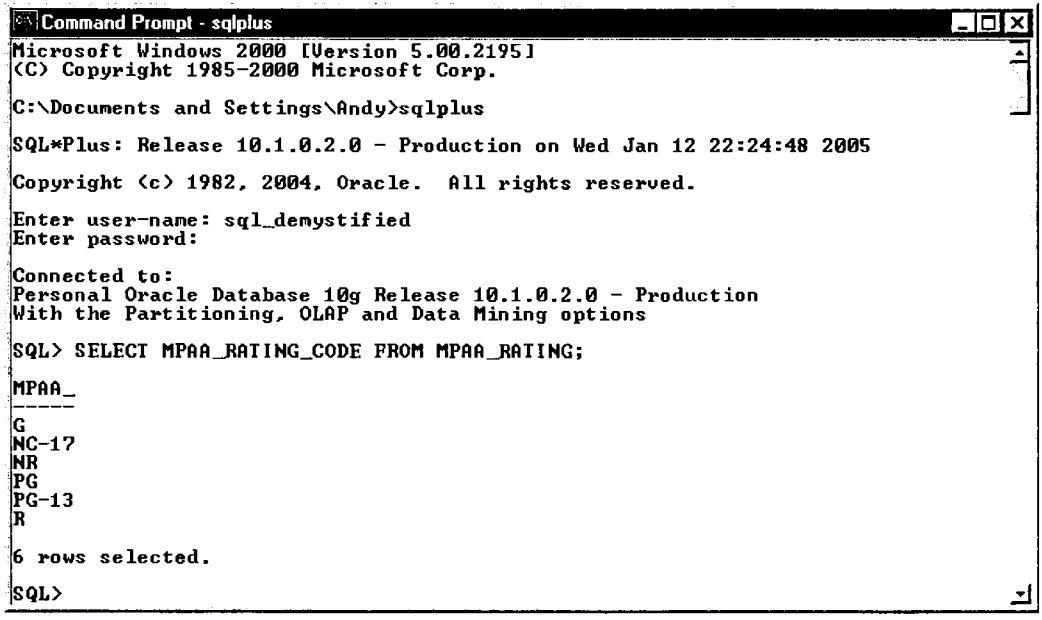


Figura 2-2 Oracle SQL*Plus rulat într-o fereastră de comandă Microsoft Windows

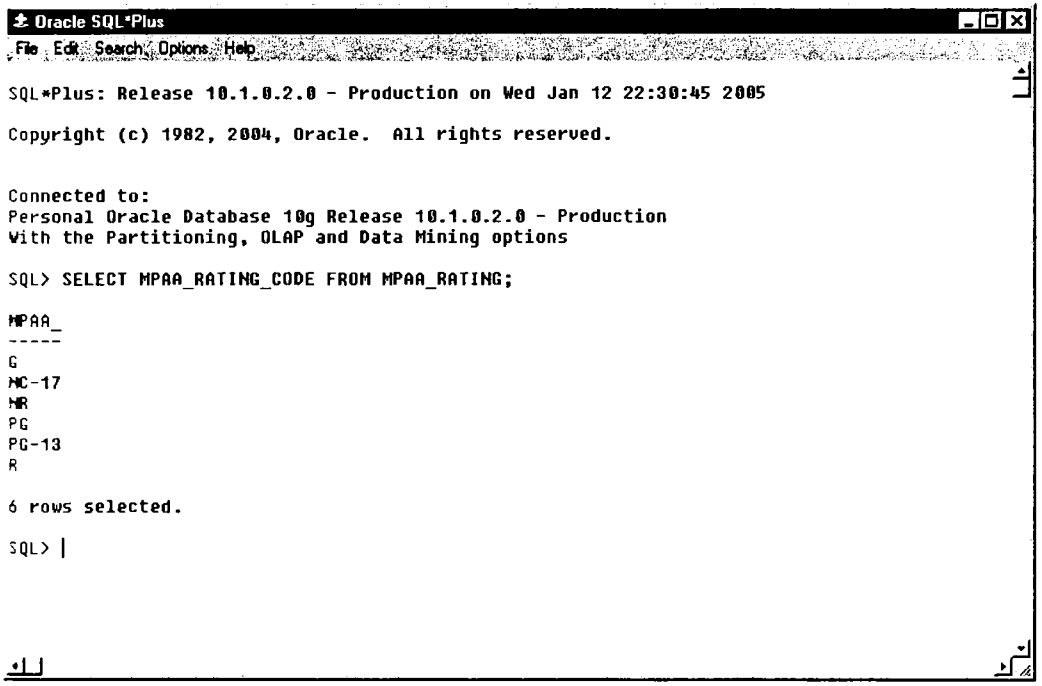


Figura 2-3 Oracle SQL*Plus rulat ca aplicație Microsoft Windows

furnizate de producătorul DBMS care sunt descărcate în fundal de browser-ul web pentru a asista în procesul de reprezentare grafică a formularelor web folosite pentru introducerea instrucțiunilor SQL și afișarea rezultatelor. Figura 2-4 prezintă clientul Oracle iSQL*Plus rulat în browser-ul Mozilla Firefox.

Tabelul următor prezintă clienți SQL oferiți de diferiți producători DBMS. Nu avem suficient spațiu în această carte pentru a prezenta toate detaliile referitoare la fiecare client SQL pe care ați putea să-l folosiți, așa că vă rog să consultați documentația producătorului DBMS pentru informații despre instalarea și utilizarea clienților SQL disponibile pentru sistemul DBMS pe care-l folosiți.

Producător	DBMS	Client SQL	Descriere
Microsoft	Access	Nu există	Microsoft Access este o bază de date de uz personal, cu clientul SQL integrat în DBMS, toate fiind rulate local pe stația de lucru a utilizatorului.
Microsoft	SQL Server	iSQL	Client SQL care rulează ca aplicație în linia de comandă într-un nucleu de comenzi Microsoft Windows.

Producător	DBMS	Client SQL	Descriere
Microsoft	SQL Server	Query Analyzer	Client SQL care rulează ca aplicație Microsoft Windows.
MySQL	MySQL	MySQL	Client SQL care rulează ca aplicație în linia de comandă sub diferite sisteme de operare, inclusiv Microsoft Windows, Linux, Mac OS X și diferite implementări Unix.
Oracle	Oracle	iSQL*Plus	Client SQL bazat pe web – acceptat în versiunile Oracle 9*i* și mai noi.
Oracle	Oracle	SQL*Plus	Client SQL care rulează ca aplicație Microsoft Windows sau ca aplicație în linia de comandă sub diferite sisteme de operare, inclusiv Microsoft Windows, Linux, Mac OS X, diferite implementări Unix și altele.
Oracle	Oracle	SQL Worksheet	Client SQL scris în Java – disponibil în Oracle 8*i* și 9*i*, dar înlocuit de iSQL*Plus în Oracle 10*g*.
Sybase	Sybase	iSQL	Clientul SQL care rulează ca aplicație în linia de comandă într-un nucleu de comenzi Microsoft Windows. Asemănările cu Microsoft SQL Server nu sunt întâmplătoare – primele versiuni Microsoft SQL Server erau bazate pe sistemul DBMS Sybase.

Un scurt istoric al limbajului SQL

Către sfârșitul anilor '70, un grup de cercetători de la IBM au dezvoltat o bază de date relațională, numită System/R, bazată pe lucrările Dr. E. F. Codd. În System/R a fost inclus un limbaj, numit SEQUEL (Structured English Query Language), pentru manipularea și extragerea datelor. Acronimul „SEQUEL" a fost ulterior condensat în abrevierea „SQL", atunci când s-a descoperit că „SEQUEL" era marcă înregistrată a companiei Hawker-Siddeley Aircraft din Marea Britanie.

Deși IBM a creat prima implementare SQL, două alte produse, cu nume diferite pentru limbajele de interogare, au fost lansate pe piață ca primele produse pentru baze de date relaționale, Oracle, furnizat de Relational Software, și INGRES, furnizat de Relational Technology. IBM a lansat în 1982 produsul SQL/DS, cu limbajul de interogare numit acum SQL (Structured Query Language). De *programarea structurată* era expresia la modă în anii '80, cuvântul „structured" din SQL nu avea nici o legătură cu programarea structurată, deoarece SQL nu este un limbaj de programare procedural. Totuși, este foarte posibil ca succesul de marketing pe care-l avea programarea structurată să fi contribuit la

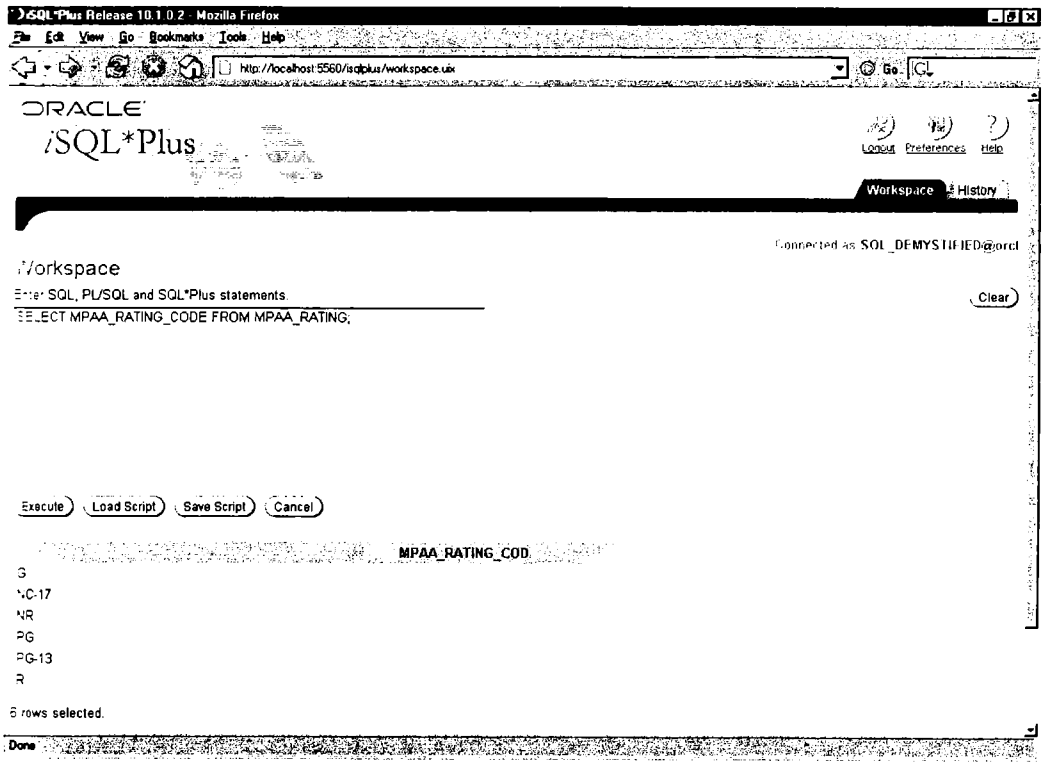


Figura 2-4 Oracle iSQL*Plus rulat în browser-ul Mozilla Firefox

favorizarea numelui „SQL" față de numele inventate de alți producători pentru propriile limbaje de interogare a datelor.

Comitetele de standardizare a limbajului SQL au fost formate de ANSI (American National Standards Institute) în 1986 și ISO (International Organization for Standardization) în 1987. Din fericire, comitetele create de cele două organizații au colaborat pentru dezvoltarea unui standard SQL comun, la nivel mondial. Doi ani mai târziu, au fost publicate primele specificații ale standardului, numite SQL-89. După trei ani, specificațiile originale au fost extinse, sub forma standardului SQL-92, care avea aproximativ 600 de pagini. A treia generație a fost numită SQL-99 sau SQL3. Cele mai multe produse RDBMS sunt construite pe baza standardului SQL-92 (numit acum SQL2). SQL3 include multe caracteristici obiectuale, necesare pentru folosirea limbajului SQL cu o bază de date relațională orientată spre obiecte, precum și extensii de limbaj care fac din SQL un limbaj de programare complet (adăugând cicluri, ramificări și construcții de comutare, de tip case). Cea mai recentă generație, numită SQL:2003, introduce caracteristici legate de XML și alte îmbunătățiri. Doar un număr foarte mic de producători au implementat componente semnificative din SQL3 și SQL:2003. Deși producătorii de sisteme DBMS au echipe care

se ocupă de respectarea standardelor, puțini dintre utilizatorii care scriu cod SQL cunosc aceste standarde. Principala cauză este faptul că aceste standarde nu sunt gratuite. Standardul SQL:2003 poate fi cumpărat de la ISO (www.iso.org) sau ANSI (webstore.ansi.org). Pentru cei care au un buget mai restrâns, este disponibilă o versiune apropiată de cea finală la Whitemarsh Information Systems Corporation (www.wiscorp.com/SQLStandards.html).

Standardele sunt importante, deoarece promovează *portabilitatea*, înțelegând prin aceasta ușurința cu care pot fi făcute să ruleze (pot fi portate) pe alte platforme. În cazul limbajului SQL, portabilitatea era slabă între produsele RDBMS ale diferiților furnizori până când aceștia au început să se conformeze standardelor publicate. Cu toate acestea, aproape toți furnizorii au adăugat extensii la „dialectul" SQL propriu, în parte deoarece doreau să diferențieze produsele proprii și în parte deoarece cerințele pieței îi forțau să implementeze caracteristici înainte a apărea standarde pentru acestea. Un astfel de exemplu este acceptarea tipurilor de date TIMESTAMP și DATE. Datele calendaristici sunt foarte importante pentru prelucrarea datelor comerciale, dar dezvoltatorii produselor RDBMS originale erau academicieni și oameni de știință, nu specialiști în prelucrări comerciale, așa că această cerință nu a fost anticipată. Ca rezultat, primele dialecte SQL nu asigurau un suport special pentru datele calendaristice. Pe măsură ce au apărut produsele comerciale, furnizorii au răspuns cererilor lansate de clienții importanți și au adăugat în grabă suportul pentru date calendaristice. Din nefericire, din cauza grabei fiecare făcut-o în felul propriu. Ori de câte ori migrați instrucțiuni SQL de la produsul unui furnizor la altul, țineți seama de diferențele dintre dialecte. Codul SQL este foarte compatibil și portabil între produsele diferiților furnizori, dar sistemele complete de baze de date pot fi rareori transferate fără anumite ajustări.

Convenții de sintaxă SQL

Această secțiune prezintă convențiile generale de sintaxă folosite pentru construirea instrucțiunilor SQL. Totuși, rețineți că există o mulțime de extensii și variații între diferiții producători. Pentru simplitate, termenul *implementare* este folosit pentru referirea fiecărei versiuni SQL a fiecărui producător (cu alte cuvinte, Oracle 9*i*, Oracle 10*g*, Microsoft SQL Server 7, Microsoft SQL Server 2000 și Microsoft SQL Server 2005 conțin *implementări* diferite ale limbajului SQL).

Convențiile de sintaxă SQL sunt mai ușor de înțeles folosind un exemplu simplu. Instrucțiunea de mai jos returnează valorile Movie ID și Movie Title pentru toate filmele din magazinul de produse video pentru care categoria MPAA este „PG":

```
SELECT MOVIE_ID, MOVIE_TITLE
   FROM MOVIE
  WHERE MPAA_RATING_CODE = 'PG';
```

Convențiile de bază sunt următoarele:

- Fiecare instrucțiune începe cu o comandă, de obicei sub forma unui singur cuvânt, care aproape întotdeauna este un verb (în limba engleză) care descrie o acțiune. În acest exemplu, instrucțiunea începe cu comanda SELECT, care este descrisă în detaliu în capitolul 4.
- Fiecare instrucțiune se termină cu un delimitator, care este, de obicei, un caracter punct și virgulă (;). Unele implementări permit schimbarea delimitatorului cu un alt caracter. Mai mult, unele implementări, cum ar fi cea din Oracle, nu execută o instrucțiune SQL căreia-i lipsește delimitatorul de sfârșit, în timp ce alte implementări consideră acest delimitator opțional.
- Instrucțiunile sunt construite într-o manieră similară cu propozițiile din limba engleză, cu unul sau mai multe spații pentru separarea elementelor de limbaj. Un *element de limbaj*, asemănător cu un cuvânt dintr-o propoziție, poate fi un cuvânt cheie (SELECT, FROM, WHERE), numele unui obiect al bazei de date (MOVIE, MOVIE_ID, MOVIE_TITLE), un operator (=) sau o constantă ('PG') care apare într-o instrucțiune.
- Instrucțiunile sunt scrise într-o formă liberă, ceea ce înseamnă că nu există reguli stricte privind poziția elementelor de limbaj pe o linie sau locul în care se poate face trecerea la o linie nouă. Totuși, în general nu este o idee bună să împărțiți un element de limbaj pe mai multe linii. Din punct de vedere logic, instrucțiunea de mai jos este identică cu cea prezentată la începutul acestei secțiuni, dar nu este la fel de ușor de citit și de înțeles:

```
SELECT MOVIE_ID,MOVIE_NAME FROM MOVIE WHERE
MPAA_RATING_CODE='PG';
```

- Instrucțiunile sunt organizate într-o serie de clauze și, de obicei, clauzele trebuie să apară într-o anumită ordine atunci când sunt folosite (multe clauze sunt opționale). În exemplul nostru, există trei clauze, fiecare începând cu un cuvânt cheie (SELECT, FROM, WHERE).
- Elementele de limbaj SQL pot fi scrise cu litere mari, cu litere mici sau în combinații. Totuși, în majoritatea implementărilor și în conformitate cu standardele ANSI/ISO, toate minusculele sunt transformate în majuscule în vederea prelucrării. Aceasta nu înseamnă că datele nu pot conține litere mici, ci că numele obiectelor din baza de date (tabele, coloane etc.) și comenzile trebuie să fie scrise cu litere mari. Excepții notabile sunt Microsoft SQL Server și Sybase, care permit modul de lucru cu diferențierea literelor mari de cele mici, caz în care numele de obiecte scrise diferit sunt tratate ca nume diferite. În MySQL, diferențierea literelor mari de cele mici în numele obiectelor este legată de capacitatea sistemului de operare de a face această diferențiere.

- Virgulele sunt folosite pentru separarea articolelor dintr-o listă. În exemplul nostru, numele a două coloane sunt specificate într-o listă separată prin virgule (MOVIE_ID, MOVIE_TITLE). Spațiile care urmează după virgule sunt opționale – puteți adăuga orice număr de spații, inclusiv zero.
- Șirurile de caractere care apar în instrucțiunile SQL trebuie să fie încadrate cu apostrofuri (unele implementări SQL permit și folosirea ghilimelelor). Constantele numerice nu sunt niciodată încadrate cu apostrofuri. Dacă în șirul de caractere trebuie să apară un caracter apostrof, sunt inserate două apostrofuri unul lângă celălalt. De exemplu, dacă vreți să găsiți în baza de date un film numit *Sophie's Choice*, veți scrie clauza WHERE astfel:

```
WHERE MOVIE_NAME = 'Sophie''s Choice'
```

- Numele obiectelor bazei de date sunt formate folosind *numai* litere, cifre și liniuțe de subliniere. Caracterul *underscore* (liniuța de subliniere) este folosit, de obicei, ca separator între cuvinte, pentru îmbunătățirea lizibilității. Așa cum am menționat anterior, unele implementări permit folosirea numelor care fac diferența între literele mari și cele mici, cum ar fi PersonMiddleName, un stil numit deseori „scriere de tip cămilă", dar acest stil nu este recomandabil dacă doriți ca instrucțiunile SQL să fie portabile pe alte implementări. În definitiv, un nume precum „PERSONMIDDLENAME" nu este foarte ușor de citit.
- În fiecare implementare SQL este definit un set de *cuvinte rezervate*, care sunt cuvinte cu o semnificație specifică pentru procesorul SQL al sistemului DBMS și, care urmare, nu trebuie folosite într-un alt context – de exemplu ca nume pentru obiectele bazei de date. Scopul acestei restricții este de a evita interpretarea greșită a instrucțiunilor SQL de către DBMS. Așa cum probabil bănuiți, lista cuvintelor rezervate diferă semnificativ de la o implementare SQL la alta, așa că este bine să consultați documentația implementării pe care o folosiți pentru a vă familiariza cu aceste cuvinte.
- Un comentariu pe o singură linie începe cu două liniuțe de despărțire (--). Cele două liniuțe se pot afla la începutul unei linii, ceea ce înseamnă că întreaga linie este considerată comentariu, sau oriunde în cadrul liniei, caz în care restul liniei, până la sfârșit, este considerat comentariu. De exemplu:

```
-- Acesta este un comentariu pe o singură linie în SQL.
```

- Un comentariu pe mai multe linii începe cu combinația dintre o diagonală la dreapta (slash) și un asterisc (/*) și continuă până la întâlnirea combinației inverse (*/). Aveți grijă să terminați corect comentariile, altfel multe linii SQL pe care le-ați scris vor fi tratate de către RDBMS ca și cum ar fi comentarii. Iată un exemplu de comentariu pe mai multe linii:

```
/*  Acesta este un comentariu pe mai multe linii.
    Continuă până la întâlnirea combinației de caractere
    care marchează sfârșitul comentariului. */
```

Categorii de instrucțiuni SQL

Instrucțiunile SQL sunt împărțite în categorii, după funcțiile pe care le îndeplinesc. Unii experți consideră aceste categorii ca fiind limbaje separate sau sublimbaje. Totuși, în SQL acestea au aceeași sintaxă de bază și respectă aceleași reguli, așa că eu le consider categorii de instrucțiuni din același limbaj. Categoriile de instrucțiuni, descrise în secțiunile următoare, sunt:

- Limbajul de definire a datelor (DDL – Data Definition Language)
- Limbajul de interogare a datelor (DQL – Data Query Language)
- Limbajul de manipulare a datelor (DML – Data Manipulation Language)
- Limbajul pentru controlul datelor (DCL – Data Control Language)
- Comenzile pentru controlul tranzacțiilor (Transaction Control Commands)

Limbajul de definire a datelor (DDL)

Limbajul de definire a datelor (DDL – Data Definition Language) include instrucțiuni SQL care permit utilizatorului bazei de date să creeze și să modifice structura obiectelor bazei de date, cum ar fi tabele, vizualizări și indexuri. Instrucțiunile SQL care folosesc comenzile CREATE, ALTER și DROP sunt considerate parte a DDL. Este important să înțelegeți că instrucțiunile DDL afectează containerele care stochează datele în baza de date, nu datele propriu-zise. Ca urmare, există instrucțiuni DDL pentru crearea, ștergerea și modificarea tabelelor, dar nici una dintre aceste instrucțiuni nu oferă posibilitatea de a crea sau modifica rânduri de date din tabelele respective. Instrucțiunile DDL sunt prezentate în capitolul 3.

Limbajul de interogare a datelor (DQL)

Limbajul de interogare a datelor (DQL – Data Query Language) include instrucțiuni SQL care permit obținerea datelor din baza de date. Deși reprezintă o parte foarte importantă a *limbajului SQL, DQL este format din instrucțiuni care folosesc o singură comandă: SELECT. Limbajul DQL este prezentat în capitolele 4, 5 și 6. Unii furnizori și autori* clasifică instrucțiunile DQL și DML în aceeași categorie.

Limbajul de manipulare a datelor (DML)

Limbajul de manipulare a datelor (DML – Data Manipulation Language) include instrucțiuni SQL care permit utilizatorului bazei de date să adauge date în baza de date (sub forma rândurilor din tabele), să șteargă date și să modifice datele existente în baza de date. Instrucțiunile SQL care folosesc comenzile INSERT, UPDATE și DELETE sunt considerate parte a DML. Instrucțiunile DML sunt prezentate în capitolul 7.

Limbajul pentru controlul datelor (DCL)

Limbajul pentru controlul datelor (DCL – Data Control Language) include instrucțiuni SQL care permit administratorilor să controleze accesul la datele din baza de date și folosirea diferitelor privilegii ale sistemului DBMS, cum ar fi privilegiul de oprire și pornire a bazei de date. Instrucțiunile SQL care folosesc comenzile GRANT și ALTER sunt considerate parte a DCL. Instrucțiunile DCL sunt prezentate în capitolul 8.

Comenzile pentru controlul tranzacțiilor

O *tranzacție în baza de date* este un set de comenzi pe care utilizatorul bazei de date vrea să le trateze ca pe o unitate funcțională de tip „totul sau nimic", înțelegând prin aceasta că întreaga tranzacție trebuie să reușească sau să eșueze. Comenzile pentru controlul tranzacțiilor (Transaction Control Commands) nu respectă cu exactitate sintaxa instrucțiunilor SQL, dar afectează puternic comportamentul instrucțiunilor SQL incluse în tranzacții. Comenzile pentru controlul tranzacțiilor sunt prezentate în capitolul 9.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. SQL
   a. Poate fi pronunțat ca literele SQL (es-q-el)
   b. Poate fi pronunțat ca și cuvântul englezesc „sequel"
   c. Poate fi folosit pentru redarea paginilor web
   d. Poate fi folosit pentru comunicarea cu orice bază de date
   e. Poate fi folosit pentru comunicarea cu bazele de date relaționale
2. SQL este
   a. Un limbaj orientat spre obiecte
   b. Un limbaj procedural
   c. Un limbaj neprocedural
   d. Un limbaj declarativ
   e. Un limbaj standard

3. Extensiile procedurale ale limbajului SQL includ
   a. Java
   b. Oracle PL/SQL
   c. C++
   d. Microsoft Transact-SQL
   e. FORTRAN
4. Într-un aranjament client/server
   a. Componentele software DBMS rulează pe server
   b. Componentele software DBMS rulează pe client
   c. Componentele software ale clientului SQL rulează pe client
   d. Componentele software ale clientului SQL rulează pe server
   e. Baza de date este stocată pe discuri conectate la client
5. Un client SQL în linia de comandă
   a. Necesită un sistem bazat pe ferestre
   b. Rulează pe o mare varietate de platforme client
   c. Necesită un browser web pe client
   d. Afișează datele și opțiunile de comandă folosind caracteristici grafice
   e. Afișează răspunsurile la comenzi sub formă de mesaje de tip text
6. Un client SQL cu interfață grafică (GUI)
   a. Necesită un sistem bazat pe ferestre
   b. Rulează pe o mare varietate de platforme client
   c. Necesită un browser web pe client
   d. Afișează datele și opțiunile de comandă folosind caracteristici grafice
   e. Afișează răspunsurile la comenzi sub formă de mesaje de tip text
7. Un client SQL bazat pe web
   a. Necesită un sistem bazat pe ferestre
   b. Rulează pe o mare varietate de platforme client
   c. Necesită un browser web pe client

d. Afișează datele și opțiunile de comandă folosind caracteristici grafice
e. Afișează răspunsurile la comenzi sub formă de mesaje de tip text

8. Clienții SQL oferiți de Oracle sunt
   a. iSQL
   b. Query Analyzer
   c. iSQL*Plus
   d. SQL*Plus
   e. SQL Worksheet
9. Clienții SQL oferiți de Microsoft sunt
   a. iSQL
   b. Query Analyzer
   c. iSQL*Plus
   d. SQL*Plus
   e. SQL Worksheet
10. Limbajul SQL a fost dezvoltat inițial
    a. De IBM
    b. De ANSI
    c. În 1982
    d. În anii '70
    e. Pe baza standardelor ANSI
11. Printre standardele SQL se numără
    a. SQL-88
    b. SQL-89
    c. SQL-92
    d. SQL-99
    e. SQL:2003
12. Extensii SQL create de furnizori
    a. Cresc portabilitatea codului SQL

b. Scad portabilitatea codului SQL
c. Contribuie la diferențierea produselor oferite de diferiți producători
d. Au fost bazate pe cererile pieței
e. Sunt compatibile între implementările diferiților producători

13. Instrucțiunile SQL
    a. Încep cu un cuvânt cheie reprezentând o comandă
    b. Se termină cu un cuvânt cheie reprezentând o comandă
    c. Încep cu un delimitator, cum ar fi caracterul punct și virgulă
    d. Se termină un delimitator, cum ar fi caracterul punct și virgulă
    e. Încep cu o paranteză deschisă
14. Elementele limbajului SQL includ
    a. Cuvinte cheie
    b. Nume ale obiectelor din baza de date
    c. Operatori
    d. Restricții
    e. Constante
15. Elementele limbajului SQL sunt separate prin
    a. Virgule
    b. Exact un spațiu
    c. Unul sau mai multe spații
    d. Linie nouă
    e. Liniuțe de subliniere
16. Numele obiectelor bazei de date pot include
    a. Paranteze
    b. Liniuțe de subliniere
    c. Numere
    d. Litere
    e. Virgule

17. Instrucțiunile SQL pot fi împărțite în următoarele categorii
    a. Limbajul de definire a datelor (DDL – Data Definition Language)
    b. Limbajul de selectare a datelor (DSL – Data Selection Language)
    c. Limbajul de replicare a datelor (DRL – Data Replication Language)
    d. Limbajul pentru controlul datelor (DCL – Data Control Language)
    e. Limbajul de manipulare a datelor (DML – Data Manipulation Language)
18. Limbajul de definire a datelor (DDL – Data Definition Language) include următoarele instrucțiuni:
    a. SELECT
    b. INSERT
    c. CREATE
    d. ALTER
    e. DELETE
19. Limbajul de interogare a datelor (DQL – Data Query Language) include următoarele instrucțiuni:
    a. SELECT
    b. INSERT
    c. CREATE
    d. ALTER
    e. DELETE
20. Limbajul de manipulare a datelor (DML – Data Manipulation Language) include următoarele instrucțiuni:
    a. SELECT
    b. INSERT
    c. CREATE
    d. ALTER
    e. DELETE

CAPITOLUL 3

Definirea obiectelor bazei de date folosind limbajul SQL

Acest capitol prezintă instrucțiunile SQL folosite pentru definirea și gestionarea obiectelor dintr-o bază de date relațională. Așa cum am menționat anterior, instrucțiunile CREATE, ALTER și DROP formează o categorie a limbajului SQL numită limbajul de definire a datelor (DDL – Data Definition Language). Începem cu prezentarea categoriei DDL deoarece trebuie să creați obiectele bazei de date înainte de a putea să inserați date în baza de date. Totuși, dacă doriți să învățați mai întâi o altă categorie de instrucțiuni SQL, cum ar fi DQL sau DML, capitolele sunt astfel scrise încât puteți începe cu oricare și să vă întoarceți ulterior la acest capitol.

Convenții de sintaxă folosite în acest capitol

Instrucțiunile SQL DDL au mai multe opțiuni decât alte instrucțiuni SQL. De aceea, am adoptat în acest capitol următoarele convenții pentru a prezenta sintaxa instrucțiunilor DDL:

- Cuvintele cheie și cuvintele rezervate din SQL sunt scrise cu majuscule, cum ar fi CREATE TABLE.
- Informațiile pe care ar trebui să le furnizați la scrierea instrucțiunilor sunt scrise cu italice, cum ar fi *nume_coloană*.
- Elementele opționale sunt încadrate în paranteze pătrate, cum ar fi [WITH TIME ZONE].
- Opțiunile dintr-o listă de elemente posibile sunt separate de o bară verticală (simbolul logic pentru „sau"), cum ar fi TABLE | VIEW | INDEX. Le veți întâlni uneori ca listă de elemente opționale, cum ar fi [NULL | NOT NULL].
- Elementele de grup care sunt explicate sau analizate ulterior pe componente (de obicei după descrierea unui tip principal de instrucțiune) sunt încadrate de caracterele „mai mic decât" și „mai mare decât", cum ar fi <specificații_pentru_coloană>.
- Un element care se poate repeta este urmat de trei puncte, cum ar fi [,<restricție_pentru_tabel>...].
- Toate celelalte simboluri, în special virgulele și parantezele, fac parte din sintaxa SQL obligatorie și, ca urmare, trebuie să fie incluse așa cum sunt scrise aici.

Tipuri de date

Înainte de a explora instrucțiunile DDL propriu-zise, trebuie să înțelegeți câteva lucruri despre modul în care sunt stocate datele în coloanele tabelelor. Așa cum vă amintiți, o *coloană* este cea mai mică unitate denumită care poate fi referită într-o bază de date relațională. Fiecare coloană trebuie să aibă asociate un nume unic și un tip de date. Un *tip de date* este o categorie pentru formatul folosit de o anumită coloană. Tipurile de date asigură câteva avantaje importante:

- Restricționarea datelor din coloana respectivă la caracterele care au sens pentru tipul de date specificat (de exemplu, cifre pentru datele numerice sau numai date calendaristice valide).
- Asigurarea unor comportamente utile pentru utilizatorul datelor. De exemplu, dacă scădeți un număr dintr-un alt număr, obțineți ca rezultat un număr; dar dacă scădeți o dată dintr-o altă dată, obțineți ca rezultat diferența în zile dintre cele două date calendaristice.

- Creșterea eficienței sistemului RDBMS la stocarea datelor din coloane. De exemplu, numerele pot fi deseori stocate ca tipuri numerice interne, ceea ce duce la economisirea spațiului, în locul stocării cifrelor ca șiruri de caractere.

SQL acceptă trei categorii de tipuri de date: tipuri predefinite, tipuri construite și tipuri definite de utilizator. *Tipurile de date predefinite* sunt cele furnizate de către producător ca parte nativă a sistemului RDBMS. *Tipurile de date construite*, cunoscute și ca *tipuri de colecții*, conțin matrice sau seturi de tipuri de date predefinite, de obicei în scopul reprezentării în RDBMS a construcțiilor de date orientate spre obiecte. *Tipurile de date definite de utilizator*, în acele sisteme RDBMS care permit folosirea acestora, permit utilizatorului bazei de date să definească propriile tipuri de date, adaptate unor scopuri specifice.

În restul acestei secțiuni sunt prezentate tipurile de date predefinite din standardul SQL:2003, urmate de o discuție asupra diferențelor dintre cele mai frecvent folosite produse RDBMS actuale. Tipurile de date construite și crearea tipurilor de date definite de utilizator sunt prea avansate pentru o carte introductivă, așa cum este aceasta.

Tipuri de date predefinite din standardul SQL:2003

Una dintre cele mai mari probleme pentru portabilitatea codului SQL între implementările diferiților producători este legată de diferențele enorme între tipurile de date acceptate. Mulți producători au creat propriile tipuri de date, ca răspuns la cererile clienților, înainte de adoptarea standardelor. Majoritatea implementărilor SQL moderne acceptă cea mai mare parte a tipurilor de date definite de standardul SQL:2003, așa cum sunt prezentate aici.

Cei mai mulți specialiști consideră convenabilă gruparea tipurilor de date SQL după clasa generală de date pe care le acceptă. Așa cum probabil bănuiți, aceste clasificări sunt discutabile și, de fapt, standardul SQL:2003 definește unele tipuri de date care aparțin mai multor grupuri. Pentru a evita confuziile inutile, am prezentat fiecare tip de date într-un singur grup, fiecare grup fiind prezentat ca secțiune separată. Pentru fiecare tip de date veți găsi o descriere, urmată de sintaxa SQL și de unul sau două exemple. Multe dintre tipurile de date sunt folosite în baza de date a magazinului de produse video, dacă doriți să vedeți mai multe exemple.

Tipuri de date pentru caractere

Tipurile de date pentru caractere conțin șiruri de caractere, adică litere, cifre și alte simboluri permise de sistemul de calcul pe care se află baza de date. În general, este bine să folosiți aceste tipuri de date pentru șirurile numerice care nu vor fi folosite niciodată în operații aritmetice sau sortări numerice, deoarece sunt mult mai flexibile decât tipurile de date numerice. De exemplu, este mai bine să stocați numerele de telefon, codurile poștale,

numerele de asigurări sociale și alte numere de acest gen ca șiruri de caractere. Tipurile de date standard pentru caractere sunt:

- **Caracter cu lungime fixă** Un șir de caractere cu lungime finită. Lungimea este specificată întotdeauna, indicând dimensiunea șirurilor de caractere stocate. Șirurile de caractere mai scurte sunt completate la dreapta cu spații, astfel încât toate șirurile de caractere dintr-o coloană de acest tip de date să aibă aceeași lungime în număr de caractere. Sintaxa SQL este:

```
CHARACTER(lungime) | CHAR(lungime)
Exemplu:   SOCIAL_SECURITY_NUMBER CHAR(9)
```

- **Caracter național** O variantă a tipului de date CHARACTER, stocată într-un set de caractere al unei anumite limbi. Acest tip de date poate fi folosit pentru traducerea șirurilor de caractere în diferite limbi. Sintaxa SQL este:

```
NATIONAL CHARACTER(lungime) | NCHAR(lungime)
Exemplu:   MOVIE_TITLE NCHAR(100)
```

- **Caracter variabil** Un șir de caractere cu lungime variabilă. Lungimea este specificată întotdeauna, indicând dimensiunea *maximă* a șirurilor de caractere stocate. Astfel puteți economisi foarte mult spațiu pentru șirurile de caractere a căror lungime variază în mod natural, cum ar fi numele și adresele. Totuși, apare o anumită suprasarcină, deoarece sistemul RDBMS trebuie să stocheze dimensiunea reală a șirului împreună cu șirul de caractere (de obicei, aceasta înseamnă un octet în plus față de lungimea fizică a șirului de caractere) și să efectueze operații suplimentare pentru calcularea lungimii șirului de caractere înainte de stocare. Sintaxa SQL este:

```
CHARACTER VARYING(lungime_maximă) | VARCHAR(lungime_maximă)
Exemplu:   CUSTOMER_NAME VARCHAR(125)
```

- **Caracter național variabil** O variantă a tipului de date pentru șiruri de caractere cu lungime variabilă, stocată într-un set de caractere al unei anumite limbi. Ca și NCHAR, acest tip de date poate fi folosit pentru traducerea șirurilor de caractere în diferite limbi. Sintaxa SQL este:

```
NATIONAL CHARACTER VARYING(lungime_maximă) | NVARCHAR(lungime_maximă)
Exemplu:   MOVIE_TITLE NVARCHAR(100)
```

Tipuri de date numerice

Așa cum sugerează și numele, tipurile de date numerice stochează *numai* numere. Acestea sunt utile mai ales pentru atributele folosite în calcule. Ca regulă generală, majoritatea proiectanților de baze de date folosesc tipuri numerice numai pentru atributele care urmează să fie folosite în calcule matematice, cum ar fi prețuri, cantități și taxe procentuale. Șirurile de caractere, deși extrem de ineficiente pentru calcule, oferă o mai mare flexibilitate în

interogările SQL, așa că sunt preferate pentru șirurile de caractere numerice care nu vor fi folosite niciodată în calcule, cum ar fi numerele de identificare a angajaților, numerele de telefon și numerele de identificare a produselor. O excepție frecvent întâlnită de la această regulă sunt coloanele pentru cheile primare, unde valorile sunt generate automat de DBMS – acestea trebuie să fie tipuri de date numerice, deoarece sistemul DBMS incrementează ultima valoare folosită pentru fiecare nou rând al tabelului.

Toate tipurile numerice au o *precizie* (un număr de cifre). De asemenea, unele tipuri numerice au și o *scală* (numărul de cifre aflate în dreapta punctului zecimal*). Tipurile întregi și tipurile numerice care includ o scală sunt numite tipuri *numerice exacte*, în timp ce numerele reale care nu includ o scală (în esență, numerele în virgulă mobilă) sunt numite tipuri *numerice aproximative*.

Tipurile numerice standard sunt:

- **Numeric** Un tip numeric exact care include o precizie și o scală. Sintaxa SQL este:

```
NUMERIC(precizie, scală)
Exemplu:   EMPLOYEE_HOURLY_RATE NUMERIC(5,2)
```

- **Zecimal** Un tip numeric exact care include o precizie și o scală. Totuși, precizia folosită de fapt de DBMS va fi egală sau mai mare decât precizia specificată. Aceasta este o diferență subtilă față de tipul NUMERIC, având scopul de a permite maparea tipului DBMS cu un tip nativ al platformei pe care rulează sistemul DBMS, care are probabil precizia și scala predefinite. (O *platformă* este o combinație între un sistem de operare și un sistem hardware, cum ar fi Windows XP pe Intel, Linux pe Intel sau Solarix pe Sun.) Sintaxa SQL este:

```
DECIMAL(precizie, scală)
Exemplu:   EMPLOYEE_HOURLY_RATE DECIMAL(5,2)
```

- **Întreg** Un tip numeric exact care include numai precizia, scris INTEGER sau INT. Numerele întregi nu au cifre zecimale, așa că scala nu este necesară, deoarece este întotdeauna zero. Standardul nu specifică valorile minime și maxime permise pentru tipurile de date întregi, așa că trebuie să verificați care sunt limitele în documentația DBMS. Sintaxa SQL este:

```
INTEGER(precizie) | INT(precizie)
Exemplu:   CUSTOMER_ACCOUNT_ID INTEGER
```

- **Întreg mic** O variantă a tipului INTEGER, scrisă SMALLINT, care stochează numere mai mici și, ca urmare, ocupă mai puțin spațiu. Standardul nu specifică

* Unele implementări RDBMS vă permit să specificați folosirea virgulei zecimale în locul punctului zecimal. Pentru compatibilitate, exemplele de cod din această carte folosesc notația cu punct zecimal, ca în limba engleză. (Nota trad.)

decât că precizia tipului SMALLINT trebuie să fie mai mică sau egală cu precizia tipului INT, așa că trebuie să consultați documentația DBMS pentru detalii. Sintaxa SQL este:

```
SMALLINT(precizie)
Exemplu:   CUSTOMER_ACCOUNT_ID SMALLINT
```

- **Întreg mare** O variantă a tipului INTEGER, scrisă BIGINT, care stochează numere mai mari și, ca urmare, ocupă mai mult spațiu. Standardul nu specifică decât că precizia tipului BIGINT trebuie să fie mai mare sau egală cu precizia tipului INT, așa că trebuie să consultați documentația DBMS pentru detalii. Sintaxa SQL este:

```
BIGINT(precizie)
Exemplu:   CUSTOMER_ACCOUNT_ID BIGINT
```

- **Număr în virgulă mobilă** Un tip numeric aproximativ, cum precizia reală mai mare sau egală cu precizia specificată. Specificarea preciziei este opțională. Pe platformele acceptate, tipul de date FLOAT este implementat, de obicei, folosind numere în virgulă mobilă. Sintaxa SQL este:

```
FLOAT(precizie)
Exemple:   INTEREST_RATE FLOAT(16)
           INTEREST_RATE FLOAT
```

- **Număr real** Un tip numeric aproximativ, cu precizie definită de implementare. Trebuie să consultați documentația DBMS pentru detalii. Sintaxa SQL este:

```
REAL
Exemplu:   INTEREST_RATE REAL
```

- **Număr real cu precizie dublă** Un tip numeric aproximativ, cu precizie definită de implementare, dar mai mare sau egală cu precizia definită pentru tipul REAL. Și în acest caz, trebuie să consultați documentația DBMS pentru detalii. Sintaxa SQL este:

```
DOUBLE PRECISION | DOUBLE
Exemplu:   INTEREST_RATE DOUBLE PRECISION
```

Tipuri de date temporale

Așa cum sugerează și numele, *tipurile de date temporale* (numite și *tipuri pentru date și ore*) stochează date care măsoară timpul, într-un mod oarecare. Așa cum am menționat anterior, standardele pentru aceste tipuri de date au rămas în urma bazelor de date comerciale, așa că majoritatea sistemelor DBMS nu se conformează în totalitate standardului. Vă recomand să consultați documentația DBMS înainte de a încerca să folosiți aceste tipuri de date.

Tipurile de date temporale conțin următoarele componente, numite de standard *câmpuri (fields)*:

Numele câmpului (cuvânt cheie SQL)	Definiție
YEAR	Anul calendaristic, pe două sau patru cifre
MONTH	Luna din an
DAY	Ziua din lună
HOUR	Ora din zi
MINUTE	Minutul din oră
SECOND	Secunda din minut
TIMEZONE_HOUR	Valoarea orei cu diferența de fus orar (zonă orară). De exemplu, Pacific Standard Time înseamnă +8 ore față de UTC (Coordinated Universal Time).
TIMEZONE_MINUTE	Valoarea minutului cu diferența de fus orar (zonă orară). De obicei este zero, dar există pe glob câteva zone orare cu diferențe de jumătate de oră.

Câmpurile TIMEZONE_HOUR și TIMEZONE_MINUTE sunt incluse în toate tipurile de date temporale pentru care este specificat cuvântul cheie WITH TIMEZONE. Tipurile de date temporale sunt:

- **Data** O dată calendaristică, incluzând câmpurile YEAR, MONTH și DAY. Sintaxa SQL este:

```
DATE [WITH TIMEZONE]
Exemplu:   DATE_ENROLLED DATE
```

- **Ora** Un tip de date pentru oră, incluzând câmpurile HOUR, MINUTE și SECOND. Sintaxa SQL este:

```
TIME [WITH TIMEZONE]
Exemplu:   TIME_ENROLLED TIME
```

- **Marcă temporală** Un tip de date combinat pentru dată și oră, incluzând câmpurile YEAR, MONTH, DAY, HOUR, MINUTE și SECOND. Sintaxa SQL este:

```
TIMESTAMP [WITH TIMEZONE]
Exemplu:   DATE_TIME_ENROLLED TIMESTAMP
```

- **Interval** Un interval de timp, incluzând câmpurile specificate printr-un *calificator de interval (interval qualifier)*, care, în esență, reprezintă precizia intervalului. Sintaxa SQL este:

```
INTERVAL câmp_de_start TO câmp_de_sfârşit | INTERVAL câmp
Exemple:   MEMBERSHIP_DURATION INTERVAL YEAR TO DAY
           TIME_WORKED  INTERVAL HOUR TO MINUTE
           RENTAL_DAYS  INTERVAL DAY
```

Tipuri de date pentru obiecte mari

Obiectele mari permit stocarea unor date care depășesc cu mult posibilitățile de stocare ale tipurilor de date prezentate până acum, ajungând deseori la dimensiuni de câțiva megaocteți. Manipularea obiectelor de dimensiuni mari este un subiect avansat care depășește scopurile acestei cărți, așa că doar vom prezenta tipurile de date definite de standard, fără detalii de sintaxă. Pentru moment, este suficient să știți că sistemele DBMS care acceptă aceste tipuri de date, de obicei, stochează datele reale separat de tabelele în care sunt referite, iar datele pot fi transferate în și din obiectele mari pe segmente, cum ar fi câțiva kiloocteți la fiecare transfer. Consultați documentația sistemului DBMS pentru elemente speciale de sintaxă asociate cu tipurile de obiecte mari.

- **Obiect mare pentru caractere** Un obiect mare pentru caractere, scris în SQL sub forma CLOB.
- **Obiect mare pentru caractere, în format național** Un obiect mare pentru caractere, stocat într-o anumită limbă, scris în SQL sub forma NLOB.
- **Obiect mare binar** Un obiect mare care conține date binare, cum ar fi o imagine sau o secvență sonoră, scris în SQL sub forma BLOB.

Un alt tip de date

Există un tip standard de date care nu este încadrat în nici una dintre categoriile prezentate anterior:

- **Boolean** Stochează o valoare logică *adevărat* sau *fals*. Sintaxa SQL este:

```
BOOLEAN
Exemplu:   PREFERRED_CUSTOMER BOOLEAN
```

Extensii ale producătorilor pentru tipuri de date și diferențe față de standard

Deviațiile și extensiile producătorilor față de standardele publicate sunt o realitate nefericită. Această secțiune evidențiază principalele diferențe între implementările SQL ale furnizorilor importanți și standardul SQL:2003. veți observa diferențe enorme în modul de implementare a tipurilor de date standard și în extensiile producătorilor față de standard. Această secțiune nu se dorește a fi o prezentare exhaustivă și, ca întotdeauna, ar trebui să consultați documentația furnizată împreună cu implementarea pe care o folosiți.

Microsoft Access

Microsoft Access este baza de date care respectă în cea mai mică măsură standardul, din toate sistemele DBMS frecvent folosite. Tipurile de date acceptate de Microsoft Access sunt:

- **Text** Echivalent cu tipul de date standard VARCHAR. Poate stoca până la 255 de caractere.
- **Memo** Conține șiruri cu cel mult 65535 de caractere, dar este definit fără specificarea unei dimensiuni.
- **Number** Echivalent cu tipul de date standard NUMERIC, dar precizia și scala sunt stabilite folosind meniul derulant Field Size (Mărimea câmpului). Numerele întregi pot fi definite alegând valoarea zero (0) pentru parametrul Decimal Places (Poziții zecimale).
- **Date/Time** Aproximativ echivalent cu tipul de date standard TIMESTAMP, dar poate stoca orice dată și oră valide între anii 100 și 9999.
- **Currency** Un tip de date numeric echivalent cu tipul NUMERIC(19,4), adică un număr cu 15 cifre în stânga punctului zecimal și cel mult 4 cifre în dreapta punctului zecimal.
- **AutoNumber** Un câmp pe 4 sau 16 octeți (în funcție de valoarea Field Size) incrementat automat cu o unitate de fiecare dată când în tabel este inserat un nou rând.
- **Yes/No** Aproximativ echivalent cu tipul de date standard BOOLEAN. Totuși, Microsoft Access permite ca acest tip de date să fie formatat ca Yes/No, On/Off sau True/False.
- **OLE Object** Similar cu tipul de date standard BLOB, acest tip de date permite stocarea unui obiect Microsoft OLE cu dimensiunea maximă de 1 GB (gigaoctet).
- **Hyperlink** Un tip de date specializat care poate conține o adresă web din Internet.
- **Lookup wizard** Un tip de date specializat care creează o legătură între o coloană din tabelul curent și conținutul unui coloane dintr-un alt tabel. Acest tip de date poate fi folosit pentru legarea dinamică a tabelelor la crearea formularelor în Microsoft Access.

Microsoft SQL Server

Limbajul Transact-SQL implementat în Microsoft SQL Server acceptă următoarele tipuri de date standard:

- **BIGINT** Numere întregi de la -2^{63} la $2^{63}-1$.
- **CHARACTER (CHAR)** Șiruri de caractere cu lungime fixă, conținând cel mult 8000 de caractere.
- **DECIMAL** Precizie și scală fixe, de la $-10^{38}+1$ la $10^{38}-1$.
- **FLOAT** Numere în virgulă mobilă în intervalul de la -1.79×10^{308} la -2.23×10^{-308}, 0 sau în intervalul de la 2.23×10^{-308} la 1.79×10^{308}.

- **INTEGER** Numere întregi de la -2^{31} la $2^{31}-1$.
- **NCHAR** Șiruri de caractere Unicode cu lungime fixă, conținând cel mult 4000 de caractere. *Unicode* este un set standard de caractere pe 16 biți folosit pentru codarea datelor, conceput și întreținut de consorțiul non-profit Unicode, Inc.
- **NUMERIC** Implementat la fel ca DECIMAL.
- **NVARCHAR** Șiruri de caractere Unicode cu lungime variabilă, conținând cel mult 4000 de caractere.
- **REAL** Numere în virgulă mobilă în intervalul de la -3.40×10^{38} la -1.18×10^{-38}, 0 sau în intervalul de la 1.18×10^{-38} la 3.40×10^{38}.
- **SMALLINT** Numere întregi de la -32768 la 32767.
- **VARCHAR** Șiruri de caractere cu lungime variabilă, conținând cel mult 8000 de caractere.

Microsoft SQL Server oferă următoarele extensii la tipurile de date standard:

- **BINARY** Date binare cu lungime fixă, având dimensiunea maximă de 8000 de octeți.
- **BIT** Date întregi cu valorile 0 sau 1.
- **DATETIME** Data și ora, de la 1 ianuarie 1753 la 31 decembrie 9999, cu o precizie de trei sutimi de secundă (3,33 de milisecunde).
- **IMAGE** Date binare cu lungime variabilă, cu dimensiunea maximă de $2^{31}-1$ octeți. Începând cu versiunea SQL Server 2005, Microsoft recomandă folosirea tipului de date VARBINARY(MAX) în locul tipului IMAGE.
- **MONEY** Valori monetare, de la -2^{63} la $2^{63}-1$, cu o precizie de 0,1 miimi din unitatea monetară.
- **NTEXT** Date Unicode cu lungime variabilă, cu dimensiunea maximă de $2^{30}-1$ caractere. Începând cu versiunea SQL Server 2005, Microsoft recomandă folosirea tipului de date NVARCHAR(MAX) în locul tipului NTEXT.
- **SMALLDATETIME** Data și ora, de la 1 ianuarie 1900 la 6 iunie 2079, cu o precizie de un minut.
- **SMALLMONEY** Valori monetare, de la -214748,3648 la 214748,3647, cu o precizie de 0,1 miimi din unitatea monetară.
- **TEXT** Date de tip text cu lungime variabilă, cu dimensiunea maximă de $2^{31}-1$ caractere. Începând cu versiunea SQL Server 2005, Microsoft recomandă folosirea tipului de date VARCHAR(MAX) în locul tipului TEXT.
- **TIMESTAMP** Un număr unic în toată baza de date, actualizat de fiecare dată când este actualizat un rând. Rețineți că acest tip de date intră în ***conflict direct*** cu tipul de date standard TIMESTAMP.
- **TINYINT** Numere întregi de la 0 la 255.

- **UNIQUEIDENTIFIER** Un identificator unic global (GUID).
- **VARBINARY** Date binare cu lungime variabilă, având dimensiunea maximă de 8000 de octeți.

Oracle

Oracle SQL acceptă următoarele tipuri de date standard:

- **BLOB** Obiecte binare mari, cu dimensiunea maximă de (4GB-1) x (dimensiunea unui bloc din baza de date).
- **CHAR** Șiruri de caractere cu lungime fixă, conținând cel mult 2000 de octeți.
- **CLOB** Obiecte de tip caracter mari, cu dimensiunea maximă de (4GB-1) x (dimensiunea unui bloc din baza de date).
- **DATE** Funcționează ca tipul standard de date DATE, dar din punct de vedere tehnic seamănă mai mult cu tipul DATETIME, deoarece poate stoca atât data, cât și ora. Acceptă date de la 1 ianuarie 4712 î.e.n. la 31 decembrie 9999 e.n. Opțional, poate fi inclusă și ora, în ore, minute și secunde. Dacă partea opțională este omisă, este stocată cu valoarea zero, echivalentă cu miezul nopții.
- **DECIMAL** Implementat ca NUMBER(*precizie,scală*).
- **DOUBLE PRECISION** Implementat ca NUMBER.
- **FLOAT** Implementat ca NUMBER.
- **INTEGER** Implementat ca NUMBER(38).
- **INTERVAL** Un interval de timp, dar sunt acceptate din standard numai variantele INTERVAL YEAR TO MONTH și INTERVAL DAY TO SECOND.
- **NCHAR** Șiruri de caractere cu lungime fixă într-o limbă națională, cu dimensiunea maximă de 2000 de octeți.
- **NCLOB** Șiruri de caractere cu lungime fixă într-o limbă națională, cu dimensiunea maximă de (4GB-1) x (dimensiunea unui bloc din baza de date).
- **NUMERIC** Implementat NUMBER(*precizie,scală*).
- **NVARCHAR** Date de tip caracter cu lungime variabilă într-o limbă națională, cu dimensiunea maximă de 4000 de octeți.
- **REAL** Implementat ca NUMBER.
- **SMALLINT** Implementat ca NUMBER(38).
- **TIMESTAMP** Data și ora în ani, luni, zile, ore, minute și secunde, dar, spre deosebire de standardul SQL:2003, precizia este exprimată în fracțiuni de secundă.
- **VARCHAR** Date de tip caracter cu lungime variabilă, conținând cel mult 4000 de caractere.

Oracle oferă următoarele extensii la tipurile de date standard:

- **BFILE** O valoare de localizare a unui fișier binar de dimensiuni mari, stocat în afara bazei de date.
- **LONG** Date de tip caracter cu lungime variabilă, cu dimensiunea maximă de 2GB. Oracle nu mai recomandă folosirea acestui tip de date, deoarece a fost înlocuit de CLOB.
- **LONG RAW** Date binare, cu dimensiunea maximă de 2GB. Oracle nu mai recomandă folosirea acestui tip de date, deoarece a fost înlocuit de BLOB.
- **NUMBER** Date numerice, cu cel mult 38 de cifre. Poate stoca valori întregi sau în virgulă mobilă (de la 1×10^{-130} la $9.99..9 \times 10^{125}$ și de la -1×10^{-130} la $9.99..99 \times 10^{125}$. Rețineți că, indiferent cât de mare sau de mic este numărul stocat, nu poate avea mai mult de 38 de cifre. Echivalent cu tipul de date standard NUMERIC.
- **NVARCHAR2** Identic cu NVARCHAR în toate versiunile până la 10*g*, dar Oracle recomandă folosirea acestui tip în locul tipului NVACHAR, deoarece este posibil ca Oracle să schimbe la un moment dat implementarea tipului NVARCHAR.
- **RAW** Date binare, cu dimensiunea maximă de 2000 de octeți. Oracle nu mai recomandă folosirea acestui tip de date, deoarece a fost înlocuit de BLOB.
- **ROWID** Șir de caractere codat Base 64*, reprezentând adresa unică a unui rând în tabelul din care face parte.
- **UROWID** Șir de caractere codat Base 64, reprezentând adresa logică a unui rând într-un tabel indexat.
- **VARCHAR2** Identic cu VARCHAR în toate versiunile de la 7 până la 10*g*, dar Oracle recomandă folosirea acestui tip în locul tipului VARCHAR, deoarece este posibil ca Oracle să schimbe la un moment dat implementarea tipului VARCHAR.

IBM DB2 Universal Database

IBM DB2 Universal Database (UDB) acceptă următoarele tipuri de date standard:

- **BLOB** Obiecte binare mari, cu dimensiunea maximă de 2.147.483.647 de octeți.

*Base64 este o tehnică de codare care reprezintă grupuri de 24 de biți din șirul de intrare ca 4 caractere tipăribile (A-Z, a-z, 0-9, + și /). Folosită, de exemplu, pentru transmiterea datelor binare pe canale care acceptă numai caractere ASCII. Base64 nu reprezintă o formă de protecție criptografică, deoarece nu folosește nici o cheie secretă. (nota trad.)

- **CHAR** Șiruri de caractere cu lungime fixă, conținând cel mult 254 de caractere.
- **CLOB** Obiecte de tip caracter mari, cu dimensiunea maximă de 2.147.483.647 de octeți.
- **DATE** O dată, conținând câmpurile standard YEAR, MONTH și DAY. Anul poate avea valori de la 0001 la 9999, iar ziua trebuie să aibă valori valide pentru anul și luna respective.
- **DECIMAL** Un număr zecimal împachetat sau pe zone, cu punct zecimal implicit și cu valori de la $-10^{31}+1$ la $10^{31}-1$.
- **DOUBLE PRECISION (DOUBLE)** Un număr în virgulă mobilă cu precizie dublă (tip de date aproximativ) cu valori în intervalul de la $-7.2x10^{75}$ la $-5.4x10^{-79}$, 0 sau în intervalul de la $+5.4x10^{-79}$ la $+7.2x10^{75}$.
- **INTEGER** Un număr întreg de la -2.147.483.648 la 2.147.483.647.
- **NUMERIC** Implementat la fel ca DECIMAL.
- **REAL** Un număr în virgulă mobilă cu precizie simplă (tip de date aproximativ) cu valori în intervalul de la $-3.4x10^{38}$ la $-1.17x10^{-37}$, 0 sau în intervalul de la $+1.17x10^{-37}$ la $+3.4x10^{38}$.
- **SMALLINT** Un număr întreg de la -32768 la 32767.
- **TIME** O valoare temporală, conținând câmpurile standard HOUR, MINUTE și SECOND, folosind un ceas de 24 de ore.
- **TIMESTAMP** Data și ora combinate, conținând câmpurile standard YEAR, MONTH, DAY, HOUR, MINUTE și SECOND. Anul poate avea valori de la 0001 la 9999, iar ziua trebuie să aibă valori valide pentru anul și luna respective. Componenta pentru ore folosește un ceas de 24 de ore.
- **VARCHAR** Date de tip caracter cu lungime variabilă, conținând cel mult 32672 de caractere.

IBM DB2 Universal Database oferă următoarele extensii la tipurile de date standard:

- **DBCLOB** Asemănător cu tipul de date NCLOB, un obiect mare format din caractere pe doi octeți, cu dimensiunea maximă de 1.073.741.823 de caractere.
- **GRAPHIC** Un șir de caractere pe doi octeți, cu lungime fixă și dimensiunea maximă de 127 de caractere.
- **VARGRAPHIC** Un șir de caractere pe doi octeți, cu lungime variabilă și dimensiunea maximă de 16536 de caractere.

MySQL

MySQL acceptă următoarele tipuri de date standard:

- **BOOLEAN** Implementat ca TINYINT(1).
- **SMALLINT** Un număr întreg mic, cu valori de la -32768 la 32767, cu semn, și de la 0 la 65535, fără semn.
- **INTEGER (INT)** Un număr întreg de la -2.147.483.648 la 2.147.483.647, cu semn, și de la 0 la 4.294.967.295, fără semn.
- **BIGINT** Un număr întreg mare, cu valori de la -9.223.372.036.854.775.808 la 9.223.372.036.854.775.807, cu semn, și de la 0 la 18.446.744.073.709.551.615, fără semn.
- **FLOAT** Un număr în virgulă mobilă cu precizie de la 0 la 24, cu valori în intervalul de la $-3.402823466 \times 10^{38}$ la $-1.175494351 \times 10^{-38}$, 0 sau în intervalul de la $+1.175494351 \times 10^{-38}$ la $+3.402823466 \times 10^{38}$.
- **DOUBLE PRECISION (DOUBLE)** Un număr în virgulă mobilă cu precizie de la 25 la 53, cu valori în intervalul de la $-1.7976931348623157 \times 10^{308}$ la $-2.2250738585072014 \times 10^{-308}$, 0 sau în intervalul de la $+2.2250738585072014 \times 10^{-308}$ la $+1.7976931348623157 \times 10^{308}$.
- **DECIMAL (DEC)** Un număr stocat ca un șir de caractere. Domeniul de valori este același ca și pentru tipul de date DOUBLE PRECISION.
- **DATE** O dată, afișată în formatul YYYY-MM-DD (AAAA-LL-ZZ). Anul poate avea valori de la 1000 la 9999.
- **TIMESTAMP** O combinație între dată și oră, cu domeniu de valori de la 01-01-1970 00:00:00 la o parte din anul 2037. MySQL stochează acest tip de date ca numărul de secunde trecut de la miezul nopții de 1 ianuarie 1970, acesta fiind modul în care sunt stocate datele față de așa-numita *epocă* Unix.
- **TIME** Ora, stocată în formatul HH:MM:SCRIPTUL. Intervalul valid pentru ore este de la -838 la +838.
- **CHAR** Un șir de caractere cu lungime fixă, conținând cel mult 255 de caractere.
- **NATIONAL CHAR (NCHAR)** Un șir de caractere cu lungime fixă, într-o limbă națională, conținând cel mult 255 de caractere.
- **VARCHAR** Un șir de caractere cu lungime variabilă, conținând cel mult 65536 de caractere.
- **NATIONAL VARCHAR (NVARCHAR)** Un șir de caractere cu lungime variabilă, într-o limbă națională, conținând cel mult 65536 de caractere.

MySQL oferă următoarele extensii la tipurile de date standard:

- **TINYINT** Un număr întreg mic, cu valori de la -128 la 127, cu semn, și de la 0 la 255, fără semn.
- **MEDIUMINT** Un număr întreg mediu, cu valori de la -8.388.608 la 8.388.607, cu semn, și de la 0 la 16.777.251, fără semn.
- **BIT** Implementat ca TINYINT(1).
- **DATETIME** Data și ora combinate, cu anul având valori de la 1000 la 9999 și folosind un ceas de 24 de ore.
- **YEAR** Anul, stocat pe două sau pe patru cifre. În formatul pe două cifre, anul poate avea valori de la 70 la 69, reprezentând anii de la 1970 la 2069. În formatul pe patru cifre, anul poate avea valori de la 1901 la 2155, plus 0000.
- **CHAR BINARY (BINARY)** Un șir de caractere cu lungime fixă, conținând cel mult 255 de octeți.
- **VARCHAR BINARY (VARBINARY)** Un șir de caractere cu lungime variabilă, conținând cel mult 65535 de octeți.
- **TINYBLOB, TINYTEXT** Un șir binar sau de caractere, conținând cel mult 255 de octeți.
- **BLOB, TEXT** Un șir binar sau de caractere, conținând cel mult 65535 de octeți.
- **MEDIUMBLOB, MEDIUMTEXT** Un șir binar sau de caractere, conținând cel mult 16.777.215 octeți.
- **LONGBLOB, LONGTEXT** Un șir binar sau de caractere, conținând cel mult 4.294.967.295 de octeți.
- **ENUM** O listă enumerare, adică o coloană de tip șir care poate conține una dintre valorile specificate în lista de valori din definiția coloanei.
- **SET** O listă set, adică o coloană de tip șir care poate conține orice număr de valori dintre valorile specificate în lista de valori din definiția coloanei.

PostgreSQL

PostgreSQL acceptă următoarele tipuri de date standard:

- **SMALLINT** Un număr întreg mic, cu valori de la -32768 la 32767.
- **INTEGER** Un număr întreg de la -2.147.483.648 la 2.147.483.647.
- **BIGINT** Un număr întreg mare, cu valori de la -9.223.372.036.854.775.808 la 9.223.372.036.854.775.807.

- **DECIMAL** Un număr exact, cu lungime variabilă, fără limită de dimensiune.
- **NUMERIC** Un număr exact, cu lungime variabilă, fără limită de dimensiune.
- **REAL** Un număr aproximativ, cu precizia de 6 cifre zecimale.
- **DOUBLE PRECISION** Un număr aproximativ, cu precizia de 15 cifre zecimale.
- **CHARACTER VARYING (VARCHAR)** Un șir de caractere cu lungime variabilă, cu limită maximă definită de utilizator și dimensiune maximă de aproximativ 1GB.
- **CHARACTER (CHAR)** Un șir de caractere cu lungime fixă și dimensiune maximă de aproximativ 1GB.
- **TIMESTAMP** O combinație între dată și oră, cu domeniu de valori de la 4713 î.e.n. la 32767 e.n. și precizie de 1 microsecundă. Poate fi inclus un parametru opțional TIMEZONE.
- **INTERVAL** Un interval de timp, cu valori pentru ani de la -178.000.000 la +178.000.000 și precizie de 1 microsecundă.
- **TIME** Ora din zi, cu precizie de 1 microsecundă. Poate fi inclus un parametru opțional TIMEZONE.
- **BOOLEAN** O valoare logică de adevărat sau false, cu diferite valori permise pentru stările adevărat și fals.

PostgreSQL oferă următoarele extensii la tipurile de date standard:

- **SERIAL** Un număr întreg incrementat automat, cu valori de la 1 la 2.147.483.647.
- **BIGSERIAL** Un număr întreg incrementat automat, cu valori de la 1 la 9.223.372.036.854.775.807.
- **MONEY** O valoare monetară, în intervalul de la -21.474.836,48 la +21.474.836,488.37.
- **TEXT** Un șir de caractere cu lungime variabilă nelimitată, dar cu dimensiunea maximă de stocare de aproximativ 1GB.
- **BYTEA** Un șir binar cu lungime variabilă.
- **Tipuri geometrice** Un set de tipuri de date care pot fi folosite pentru reprezentarea obiectelor bidimensionale, inclusiv POINT, LINE, LSEG, BOX, PATH, POLYGON și CIRCLE.
- **Tipuri pentru adrese de rețea** Un set de tipuri de date care pot fi folosite pentru stocarea adreselor IPv4, IPv6 și MAC.
- **BIT** Un șir cu lungime fixă, conținând numai valori binare 1 și 0.

- **BIT VARYING** Un șir cu lungime variabilă, conținând numai valori binare 1 și 0.
- **Matrice** O matrice de tipuri de date predefinite sau definite de utilizator.
- **Tipuri compuse** Tipuri de date definite de utilizator, compuse din alte tipuri de date.
- **Tipuri pentru identificarea obiectelor** Un set de tipuri de date folosite pentru identificarea internă a obiectelor din baza de date.

Valori NULL și logica bazată pe trei valori

La definirea coloanelor din tabelele bazei de date, aveți opțiunea de a specifica dacă valorile nule (null) sunt permise în coloana respectivă. O *valoare nulă* într-o bază de date relațională este un cod special care poate fi plasat pe o coloană pentru a indica faptul că valoarea pentru coloana respectivă nu este cunoscută. O valoare nulă nu este același lucru cu un spațiu, un șir vid sau valoarea zero – este un cod special care nu are nici o altă semnificație în baza de date.

O modalitate uniformă de tratare a valorilor nule este un standard ANSI/ISO pentru bazele de date relaționale. Totuși, au avut loc multe dezbateri contradictorii privind utilitatea acestei opțiuni, în special deoarece baza de date nu vă poate spune *de ce* este necunoscută valoarea respectivă. De exemplu, dacă lăsați necompletată valoarea MPAA Rating Code pentru un film din tabelul MOVIE al bazei de date pentru magazinul de produse video, nu știți dacă valoarea este necunoscută deoarece filmul încă nu a fost clasificat, sistemul de clasificare nu se aplică filmului respectiv dintr-un motiv oarecare sau filmul a fost deja clasificat, dar încă nu știți în ce categorie. O altă problemă este faptul că valorile nule nu sunt egale cu nimic, nici măcar cu alte valori nule, ceea ce introduce o logică bazată pe trei valori la căutările în baza de date. Dacă sunt folosite valori nule, o căutare poate returna condiția *adevărat* (valoarea coloanei corespunde criteriilor de căutare), *fals* (valoarea coloanei nu corespunde criteriilor de căutare) sau *necunoscut* (valoarea coloanei este nulă). Dezvoltatorii care scriu aplicațiile trebuie să trateze valorile nule ca pe un caz special.

În Microsoft Access, restricția NOT NULL este controlată de opțiunea Required din panoul de proiectare a tabelului. În SQL DDL, specificați cuvintele cheie NULL sau NOT NULL în definiția coloanei (în dreapta tipului de date al coloanei). Atenție la valorile prestabilite! În Oracle, DB2 și majoritatea celorlalte baze de date relaționale, dacă omiteți specificarea acestei restricții, valoarea prestabilită este NULL, ceea ce înseamnă că în coloana respectivă puteți avea valori nule. Pe de altă parte, în Microsoft SQL Server și Sybase Adaptive Server este exact invers: dacă omiteți specificarea acestei restricții, valoarea prestabilită este NOT NULL, ceea ce înseamnă că în coloana respectivă *nu puteți avea* valori nule.

Instrucțiuni DDL (Data Definition Language)

Instrucțiunile DDL (Data Definition Language) definesc obiectele bazei de date, dar nu inserează și nu actualizează date în obiectele respective (această funcție fiind îndeplinită de instrucțiunile DML). În SQL, există trei comenzi de bază pentru instrucțiunile DDL:

- **CREATE** Creează în baza de date un nou obiect, de tipul specificat în instrucțiune. Deoarece sintaxa este diferită, vom prezenta separat instrucțiunile CREATE DATABASE, CREATE TABLE, CREATE INDEX și CREATE VIEW.
- **ALTER** Modifică definiția unui obiect existent în baza de date, de tipul specificat în instrucțiune. Deoarece este o frecvent folosită, vom prezenta în acest capitol instrucțiunea ALTER TABLE. Consultați documentația DBMS pentru mai multe informații despre instrucțiunile ALTER DATABASE, ALTER SYSTEM, ALTER USER, ALTER SESSION și așa mai departe.
- **DROP** Șterge (distruge) un obiect existent în baza de date, de tipul specificat în instrucțiune.

Instrucțiunea CREATE DATABASE

Așa cum am menționat în capitolul 1, definirea unei baze de date diferă destul de mult de la o implementare la alta. Totuși, în general, majoritatea sistemelor DBMS vă cer să creați baza de date înainte de a putea să creați obiectele din baza de date. Sintaxa generală pentru instrucțiunea CREATE DATABASE este

```
CREATE DATABASE nume_bazadedate [opțiuni_specifice_producătorului]
```

În Oracle, instrucțiunea CREATE USER creează în baza de date o schemă, care este aproximativ echivalentă cu ceea ce alte sisteme DBMS numesc o bază de date.

Standardul SQL prevede și o instrucțiune CREATE SCHEMA, care permite crearea unor grupuri de obiecte din baza de date, pentru simplificarea administrării, dar veți descoperi diferențe foarte mari în implementarea acestei instrucțiuni de la un producător la altul, în special deoarece fiecare producător implementează fizic baza de date într-un mod foarte diferit.

Instrucțiunea CREATE TABLE

CREATE TABLE este una din instrucțiunile fundamentale din SQL. Modelul relațional cere ca toate datele stocate să fie ancorate într-un tabel, așa că posibilitatea de a stoca orice într-o bază de date începe întotdeauna cu crearea unui tabel. Sintaxa de bază pentru instrucțiunea CREATE TABLE este

```
CREATE TABLE nume_tabel
  ( <definiţie_coloană>
    [,<definiţie_coloană> ...])
    [,<restricţie_tabel>...];
```

Sintaxa pare destul de simplă și, exceptând complicațiile legate de definirea coloanelor, instrucțiunea CREATE TABLE chiar este simplă. Fiecare instrucțiune include numele tabelului și o listă cu una sau mai multe definiții de coloane, separate prin virgule și încadrate între paranteze. Numele tabelului trebuie să fie unic în baza de date și să respecte convențiile de sintaxă SQL descrise în capitolul 2. Fiecare tabel trebuie să aibă cel puțin o coloană – ceea ce este logic, dacă vă gândiți la modul în care este construit un tabel. Unele implementări permit specificarea unor atribute de stocare fizică în instrucțiunile CREATE TABLE, cum ar fi spațiul alocat tabelului, așa că este bine să consultați documentația DBMS pentru mai multe informații. În continuare vom discuta despre definirea coloanelor, apoi despre restricțiile pentru tabele.

Definirea coloanelor în SQL DDL

Sintaxa de bază folosită pentru definirea coloanelor unui tabel este:

```
<definiţie_coloana>:
   nume_coloană  tip_de_date
   [DEFAULT expresie]
   [NULL | NOT NULL]
   [<restricţie_coloană>]
```

Ca exemplu, iată instrucțiunea DDL pentru crearea tabelului CUSTOMER_ACCOUNT din baza de date a magazinului de produse video:

```
CREATE TABLE CUSTOMER_ACCOUNT (
CUSTOMER_ACCOUNT_ID INTEGER    NOT NULL,
CUSTOMER_HOLD_INDIC CHAR(1) DEFAULT 'N' NOT NULL
   CHECK (CUSTOMER_HOLD_INDIC IN ('Y','N')),
DATE_ENROLLED       DATE     NOT NULL,
DATE_TERMINATED     DATE     NULL,
CUSTOMER_DEPOSIT_AMOUNT NUMERIC(5,2) NULL,
CREDIT_CARD_ON_FILE_INDIC CHAR(1)  NOT NULL
   CHECK (CREDIT_CARD_ON_FILE_INDIC IN 'Y','N')),
CHILD_RENTAL_ALLOWED_INDIC CHAR(1)  NOT NULL
   CHECK (CHILD_RENTAL_ALLOWED_INDIC IN ('Y','N')),
   PRIMARY KEY (CUSTOMER_ACCOUNT_ID));
```

Să aruncăm o privire asupra componentelor din definiția unei coloane:

- **Numele coloanei** Numele coloanei trebuie să fie unic în cadrul tabelului și trebuie să respecte convențiile de sintaxă SQL descrise în capitolul 2.
- **Tipul de date** Tipul de date trebuie să fie un tip de date valid pentru implementarea DBMS, conform descrierilor din acest capitol.
- **Restricțiile coloanei** Sunt prezentate în secțiunea următoare.

Restricțiile coloanelor

Restricțiile unei coloane limitează (constrâng) într-un mod oarecare valorile ce pot fi stocate într-o coloană a unui tabel. (Restricțiile au fost prezentate în capitolul 1.) Din punct de vedere tehnic, clauzele DEFAULT și NULL | NOT NULL sunt forme speciale de restricții, dar acestea nu sunt implementate întotdeauna în același mod în toate sistemele DBMS. Restricția unei coloane se poate referi la o singură coloană, dar există o modalitate simplă de ocolire, deoarece orice restricție de coloană poate fi rescrisă ca restricție de tabel. (Restricțiile tabelelor sunt prezentate mai târziu în acest capitol.) Restricțiile coloanelor pot avea oricare dintre următoarele forme:

- **Clauza DEFAULT** O expresie care este aplicată coloanei atunci când în tabel este inserat un nou rând, care nu conține o valoare explicită pentru coloana respectivă. Expresia poate fi orice expresie validă care poate fi interpretată de SQL, cum ar fi o constantă, o funcție SQL sau o altă sintaxă care, în urma evaluării de către motorul SQL din DBMS, produce o valoare corespunzătoare pentru coloana respectivă. Ca exemplu, observați clauza DEFAULT 'N' pentru coloana CUSTOMER_HOLD_INDIC din tabelul CUSTOMER_ACCOUNT. Prin specificarea acestei valori prestabilite vă asigurați că orice cont de client nou creat va primi întotdeauna valoarea 'N' (contul nu este blocat) dacă instrucțiunea care inserează noul rând nu furnizează o valoare pentru această coloană sau folosește cuvântul cheie DEFAULT pentru valoarea coloanei. O altă utilizare obișnuită a clauzei DEFAULT este inserarea unor date, cum ar fi stocarea datei curente pentru efectuarea unei tranzacții, la inserarea noilor rânduri în baza de date. Totuși, sintaxa pentru adăugarea datei curente variază foarte mult de la o implementare DBMS la alta, așa că am evitat să o folosesc în instrucțiunile DDL pentru baza de date a magazinului de produse video. Iată sintaxa SQL și un exemplu de coloană cu o clauză DEFAULT:

```
[DEFAULT expresie]
Exemplu:
CUSTOMER_HOLD_INDIC CHAR(1) DEFAULT 'N' NOT NULL
```

- **Restricția NULL | NOT NULL** Așa cum am discutat mai devreme în acest capitol, specificarea cuvântului cheie NULL permite stocarea valorilor nule într-o coloană, în timp ce NOT NULL nu permite stocarea valorilor nule în coloana respectivă. *Atenție la valorile prestabilite!* Dacă omiteți această clauză în bazele de date Microsoft SQL Server și Sybase, sistemul DBMS va considera prestabilită clauza NOT NULL, dar dacă omiteți clauza în majoritatea celorlalte baze de date, cum ar fi Oracle, DB2, MySQL și PostgreSQL, se întâmplă exact invers: sistemul DBMS va considera prestabilită clauza NULL. O restricție NOT NULL poate fi scrisă și sub forma unei restricții CHECK (așa cum vom vedea în continuare), cu condiția IS NOT NULL. Iată sintaxa SQL și câteva exemple:

```
NULL | NOT NULL
Exemple:
DATE_ENROLLED DATE NOT NULL
DATE_TERMINATED DATE NULL
```

- **Restricția CHECK** O restricție de verificare (check) poate fi folosită pentru impunerea unei reguli care poate fi aplicată unei singure coloane a unui tabel. Condiția inclusă în restricție trebuie să fie îndeplinită ori de câte ori datele din coloana respectivă a tabelului sunt modificate – în caz contrar, sistemul DBMS va respinge modificarea și va afișa un mesaj de eroare. Condiția din restricția CHECK a unei coloane nu poate referi nici o altă coloană a tabelului. Observați că în exemplul tabelului CUSTOMER_ACCOUNT există trei coloane (CUSTOMER_HOLD_INDIC, CREDIT_CARD_ON_FILE_INDIC și CHILD_RENTAL_ALLOWED_INDIC) cu restricții CHECK care permit stocarea în coloanele respective numai a valorilor 'Y' și 'N'. Opțional, restricțiile coloanelor pot fi denumite, prin specificarea unui nume unic după cuvântul cheie CONSTRAINT, ceea ce reprezintă o practică bună de programare, deoarece numele restricției apare, de obicei, în orice mesaj de eroare afișat la încălcarea restricției. Iată sintaxa restricției CHECK și un exemplu:

```
[CONSTRAINT nume_restricţie] CHECK (condiţie)
Exemplu:
CREDIT_CARD_ON_FILE_INDIC CHAR(1) NOT NULL
  CHECK (CREDIT_CARD_ON_FILE_INDIC IN 'Y','N'))
```

Așa cum am menționat deja, o clauză NOT NULL, precum cea folosită pentru coloana CUSTOMER_ACCOUNT_ID, poate fi rescrisă și sub forma unei restricții CHECK. Iată cum ar arăta definiția coloanei cu o restricție CHECK care include un nume opțional pentru restricție:

```
CUSTOMER_ACCOUNT_ID INTEGER
  CONSTRAINT CK_CUST_ACCT_ID CHECK (ACCOUNT_ID IS NOT NULL)
```

- **Restricția UNIQUE** O restricție UNIQUE impusă asupra unei coloane garantează unicitatea valorilor din coloana respectivă a tabelului, de obicei cu ajutorul unui index creat automat de DBMS. Iată sintaxa restricției de unicitate și un exemplu de utilizare:

```
[CONSTRAINT nume_restricţie] UNIQUE
Exemplu:
CUSTOMER_ACCOUNT_ID INTEGER NOT NULL UNIQUE
```

- **Restricția PRIMARY KEY** O restricție de cheie primară (PRIMARY KEY) impusă asupra unei coloane declară coloana respectivă ca fiind cheia primară a tabelului, ceea ce înseamnă că în coloana respectivă nu pot exista valori nule, iar valorile trebuie să fie unice în cadrul tabelului. Ca și în cazul restricțiilor de unicitate, majoritatea sistemelor DBMS impun restricția de cheie primară cu ajutorul unui index creat automat. Iată sintaxa restricției de cheie primară și un exemplu de utilizare:

```
[CONSTRAINT nume_restricţie] PRIMARY KEY
Exemplu:
CUSTOMER_ACCOUNT_ID INTEGER NOT NULL PRIMARY KEY
```

- **Restricția referențială (FOREIGN KEY)** O restricție referențială impusă asupra unei coloane (numită uneori și restricție de cheie externă) definește relația dintre o cheie externă și o cheie primară, astfel încât sistemul DBMS să poată garanta că valoarea, dacă nu este nulă, referă întotdeauna valoarea unei chei primare existente. Iată sintaxa restricției referențiale:

```
[CONSTRAINT nume_restricţie]
  REFERENCES nume_tabel(nume_coloană)
 [ON DELETE CASCADE | ON DELETE SET NULL]
Exemplu:
MPAA_RATING_CODE CHAR(5) NOT NULL
  REFERENCES MPAA_RATING (MPAA_RATING_CODE)
```

Clauza opțională ON DELETE spune sistemului DBMS ce să facă atunci când este șters rândul referit din tabelul părinte (rândul care conține cheia primară corespondentă), cu opțiunea de a șterge toate rândurile care conțin cheia externă (CASCADE) sau de a insera valori nule pentru toate cheile externe (SET NULL). Rețineți că majoritatea sistemelor DBMS, dar nu toate, acceptă clauza ON DELETE.

Restricțiile tabelelor

Așa cum am menționat anterior, restricția unei coloane poate fi rescrisă și ca restricție a întregului tabel, astfel încât clauza care definește restricția să apară în instrucțiunea CREATE

TABLE după definițiile tuturor coloanelor, nu după definiția unei coloane. Principalul avantaj al restricțiilor la nivelul tabelului este că pot referi mai multe coloane. Semnificația fiecărei restricții a fost deja discutată (în secțiunea despre restricțiile coloanelor) așa că aici vom prezenta numai sintaxa generală și câteva exemple. Toate exemplele folosesc tabelul CUSTOMER_ACCOUNT, dar unele au fost modificate, astfel încât să ilustreze ideile principale prezentate.

- **Restricția CHECK**

```
[CONSTRAINT nume_restricţie] CHECK (condiţie)
Exemplu:
CONSTRAINT CK_CUSTOMER_DEPOSIT_AMOUNT
   CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR
         CUSTOMER_DEPOSIT_AMOUNT IS NULL)
```

Restricția din exemplul de mai sus împiedică stocarea unei valori negative în coloana CUSTOMER_DEPOSIT_AMOUNT. Observați operatorul SAU, folosit pentru a permite stocarea valorilor nule în coloană. Dacă nu ar fi fost inclusă această condiție, coloana nu ar fi acceptat valori nule, deoarece o valoare nulă nu este mai mare sau egală cu zero.

- **Restricția UNIQUE**

```
[CONSTRAINT nume_restricţie] UNIQUE (nume_coloană [,nume_coloană...])
Exemplu:
CONSTRAINT UK_CUST_ACCT_DATE_ENROLLED
   UNIQUE (CUSTOMER_ACCOUNT_ID, DATE_ENROLLED)
```

Conform acestei restricții, combinația de coloane CUSTOMER_ACCOUNT_ID și DATE_ENROLLED trebuie să fie unică în rândurile din tabelul CUSTOMER_ACCOUNT. În acest exemplu, coloana CUSTOMER_ACCOUNT_ID este oricum unică, așa că impunerea noii restricții nu prea are sens, dar am inclus-o aici pentru a ilustra folosirea unei restricții de unicitate bazate pe mai multe coloane.

- **Restricția PRIMARY KEY**

```
[CONSTRAINT nume_restricţie]
   PRIMARY KEY (nume_coloană [,nume_coloană...])
Exemplu:
CONSTRAINT PK_CUSTOMER_ACCOUNT
   PRIMARY KEY (CUSTOMER_ACCOUNT_ID)
```

Restricția de mai sus este chiar definiția cheii primare din tabelul CUSTOMER_ACCOUNT folosit ca exemplu în această secțiune, dar în acest caz am denumit explicit restricția.

- **Restricția referențială (FOREIGN KEY)**

```
[CONSTRAINT nume_restricție]
  FOREIGN KEY (nume_coloană [,nume_coloană...])
  REFERENCES nume_tabel(nume_coloană [,nume_coloană...])
[ON DELETE CASCADE | ON DELETE SET NULL]
```

Observați că, spre deosebire de forma pentru coloane a restricției referențiale, aceasta poate referi coloane multiple. Așa cum a fost proiectat, tabelul CUSTOMER_ACCOUNT nu are coloane chei externe, dar să ne gândim la un alt model. Unii proiectanți de baze de date nu sunt de acord cu folosirea restricțiilor CHECK pentru a controla valorile din coloane, deoarece adăugarea sau eliminarea valorilor implică modificarea proiectului bazei de date. Să presupunem, de exemplu, că la o îmbunătățire ulterioară a bazei de date pentru magazinul de produse video trebuie să permitem stocarea unei noi valori, 'E' (de la „exceptat"), în coloana CREDIT_CARD_ON_FILE_INDIC. Puteți modifica restricția CHECK pentru a permite stocarea noii valori în tabel, dar dacă ați fi pus toate valorile și descrierile într-un tabel separat (deseori numit tabel „de coduri", „referință" sau „de căutare"), ar fi fost suficient să adăugați noul cod în tabelul respectiv, fără nici o modificare asupra proiectului bazei de date. Tocmai pentru obținerea acestei flexibilități, baza de date a magazinului de produse video conține tabele pentru informații precum MPAA_RATING_CODE – dacă asociația MPAA își schimbă sistemul de clasificare, este suficient să modificați datele din tabelul de coduri. De asemenea, tabelele de coduri sunt o sursă elegantă pentru popularea cu valori a listelor derulante din componente de aplicație, precum formularele din paginile web.

Presupunând că a fost creat un tabel numit CARD_ON_FILE_TYPE, cu o cheie primară numită CARD_ON_FILE_CODE, iată cum arată restricția referențială care definește coloana CREDIT_CARD_ON_FILE_INDIC ca fiind cheie externă:

```
CONSTRAINT FK_CARD_ON_FILE_INDIC
   FOREIGN KEY (CREDIT_CARD_ON_FILE_INDIC)
   REFERENCES CARD_ON_FILE_TYPE (CARD_ON_FILE_CODE)
```

Așa cum puteți vedea, nu întotdeauna există un singur mod „corect" de proiectare a unei baze de date, ci mai multe soluții din care puteți alege. Altfel spus, proiectarea bazelor de date nu este o știință exactă. În general, este recomandabil să dați cheii externe același nume ca și cheii primare, dar așa cum puteți vedea din acest exemplu, SQL vă permite să folosiți un alt nume, dacă așa vreți sau așa trebuie.

Instrucțiunea CREATE INDEX

Instrucțiunea CREATE INDEX este mult mai simplă decât instrucțiunea CREATE TABLE despre care tocmai am discutat. Iată sintaxa de bază a acestei instrucțiuni:

```
CREATE [UNIQUE] INDEX nume_index ON nume_tabel
   (nume_coloană [,nume_coloană [ASC | DESC]...]);
```

- Cuvântul cheie opțional UNIQUE definește indexul ca unic, însemnând că nu pot exista două rânduri din tabel cu exact aceeași combinație de valori în coloanele specificate. Probabil vă întrebați de ce ați dori să faceți acest lucru, dacă puteți defini o restricție de unicitate, cu același rezultat. Răspunsul este că alegerea uneia dintre cele două soluții ține de stilul și opțiunile personale. Unii proiectanți nu vor ca sistemul DBMS să creeze automat un index (așa cum se întâmplă în cazul restricțiilor de unicitate), deoarece crearea explicită a unui index le oferă un nivel mai ridicat de control. Alți proiectanți preferă să lase sistemul DBMS să gestioneze crearea indexurilor, nu numai fiindcă astfel au mai puțin de lucru, ci și fiindcă această metodă este mai puțin predispusă la erori și duce la o mai bună consecvență a bazei de date.
- Cuvântul cheie opțional ASC creează indexul în ordine crescătoare, în timp ce DESC creează indexul în ordine descrescătoare. Dacă nu este specificată nici una dintre cele două opțiuni, ordinea prestabilită este crescătoare.
- Un index trebuie să aibă cel puțin o coloană, dar, practic, nu există o limită superioară a numărului de coloane.

Indexurile sunt instrumente puternice, deoarece permit sistemului DBMS să găsească datele mult mai repede, așa cum indexul unei cărți vă permite să găsiți rapid ceea ce vă interesează. Mai mult, indexurile pe coloanele cheilor externe cresc mult performanțele la unirea tabelelor. Totuși, indexurile ocupă spațiu de stocare și trebuie să fie întreținute – de fiecare dată când se modifică o valoare de pe o coloană referită de un index, trebuie să fie modificat și indexul corespunzător. Sistemul DBMS întreține automat indexul, dar activitatea de întreținere consumă din resursele calculatorului.

Instrucțiunea CREATE VIEW

Așa cum am menționat în capitolul 1, vizualizările oferă mari avantaje utilizatorilor unei baze de date, deoarece permit ajustarea datelor afișate în funcție de cerințele individuale și maschează operațiile complexe. Atunci când sunt create corect, vizualizările implică suprasarcini minime și nu stochează date. În esență, o vizualizare este o interogare SQL stocată, care poate fi referită de instrucțiunile SQL DML și DQL ca și cum ar fi un tabel real. Unii consideră că vizualizările sunt „tabele virtuale", deoarece se comportă la fel ca tabelele (cu unele restricții), dar nu există ca tabele fizice. Sintaxa generală a instrucțiunii CREATE VIEW este

```
CREATE [OR REPLACE] VIEW nume_vizualizare AS interogare_sql;
```

- Cuvântul cheie opțional OR REPLACE elimină necesitatea de a șterge o vizualizare existentă înainte de a o crea din nou. Dacă specificați parametrul

opțional OR REPLACE și vizualizarea există deja, este înlocuită, iar dacă nu există, noua vizualizare este adăugată în baza de date.

- Numele vizualizării trebuie să respecte aceleași reguli de denumire ca și tabelele și alte obiecte ale bazei de date. Așa cum veți învăța în capitolul 4, în interogările SQL sunt specificate numele obiectelor din care sunt selectate datele, dar nu și tipul acestor obiecte. Ca urmare, numele vizualizărilor trebuie să fie unice pentru toate tabelele și vizualizările din baza de date. Cu alte cuvinte, numele vizualizărilor și numele tabelelor trebuie să provină din același *spațiu de nume*, adică același domeniu de nume.
- Interogarea SQL inclusă în definiția vizualizării poate fi orice instrucțiunea SQL SELECT validă. Veți învăța despre această instrucțiune SQL esențială în capitolul 4. Crearea vizualizărilor urmează o cale naturală de evoluție – lucrați cu interogarea SQL, faceți modificările necesare și rulați din nou interogarea până când obțineți rezultatele dorite. Apoi adăugați instrucțiunea CREATE VIEW în fața interogării cu care ați lucrat și rulați instrucțiunea pentru a stoca permanent interogarea în baza de date, ca vizualizare. Aceasta este o modalitate foarte productivă (și plăcută) de a lucra cu bazele de date.

Instrucțiunea ALTER TABLE

După ce creați un tabel, aproape orice ați specificat în instrucțiunea CREATE TABLE poate fi modificat folosind instrucțiunea ALTER TABLE. În ultimii ani, producătorii de sisteme DBMS au oferit o flexibilitate din ce în ce mai mare în privința modificărilor ce pot fi făcute asupra tabelelor existente (adică fără ștergerea și recrearea tabelelor). În mare parte, această flexibilitate a fost impusă de necesitate, din cauza cerinței de a menține bazele de date permanent disponibile (cu timp de rulare 7 x 24) și a ratei mari de acumulare a datelor, care au dus la obținerea unor tabele atât de mari, încât nu mai este practică ștergerea și recrearea lor.

Utilizarea instrucțiunii ALTER TABLE este un alt domeniu în care au un rol important stilul și preferințele personale. Mulți administratori de baze de date preferă să folosească instrucțiuni CREATE TABLE cât mai simple, evitând să definească restricții în instrucțiunile CREATE TABLE. Aceștia adaugă după instrucțiunea CREATE TABLE instrucțiuni ALTER TABLE prin care specifică toate restricțiile necesare (cheie primară, cheie externă, unicitate, verificare și așa mai departe). Dezavantajul acestei metode este acela că necesită scrierea unei cantități mai mari de cod. Pe de altă parte, instrucțiunea CREATE TABLE este mult mai ușor de înțeles fără restricții, iar scrierea separată a restricțiilor simplifică refolosirea instrucțiunilor, dacă este nevoie să eliminați și apoi să creați din nou restricțiile.

Deși există unele diferențe între implementările DBMS, iată o listă cu tipurile de modificări acceptate, de obicei, de instrucțiunea ALTER TABLE, împreună cu sintaxa generală pentru fiecare tip:

- Adăugarea unei coloane la un tabel. Definirea coloanei se face cu aceeași sintaxă ca și în cazul instrucțiunii CREATE TABLE.

```
ALTER TABLE nume_tabel
 ADD ( <definiţie_coloană>
     [,<definiţie_coloană> ...]);
Exemplu:
ALTER TABLE CUSTOMER_ACCOUNT
 ADD (CUSTOMER_HOLD_DATE   DATE NULL,
       HOLD_PLACED_BY      VARCHAR(50));
```

- Modificarea definiției unei coloane. Majoritatea sistemelor DBMS nu vă permit să scădeți precizia unei coloane dacă tabelul conține date și foarte puține vă permit să schimbați tipul de date al unei coloane existente. Totuși, în general sunt acceptate creșterea preciziei unei coloane, adăugarea sau modificarea valorii prestabilite pentru o coloană și trecerea de la NULL la NOT NULL sau invers. Modificarea restricțiilor fără nume, definite pentru coloane, este puțin problematică – un alt argument în favoarea denumirii tuturor restricțiilor.

```
ALTER TABLE nume_tabel
 MODIFY [COLUMN] (<definiţie_coloană>
                  [,<definiţie_coloană> ...]);
Exemplu:
ALTER TABLE CUSTOMER_ACCOUNT
 MODIFY (CUSTOMER_DEPOSIT_AMOUNT   NUMERIC(7,2)
         DEFAULT 0 NOT NULL);
```

NOTĂ: Microsoft SQL Server, DB2 și PostgreSQL folosesc cuvântul cheie ALTER COLUMN în loc de MODIFY COLUMN, în timp ce MySQL și Oracle folosesc cuvântul cheie MODIFY, ca în exemplul de mai sus. De asemenea, rețineți că majoritatea sistemelor DBMS nu vă permit să specificați clauza NOT NULL pentru o coloană dacă în tabel există rânduri care au valori nule în coloana respectivă – trebuie ca mai întâi să înlocuiți valorile nule cu altceva. Instrucțiunea UPDATE, necesară pentru a modifica valorile datelor dintr-o coloană, este prezentată în capitolul 7.

- Adăugarea unei restricții. Definiția restricției este identică cu definiția unei restricții care ar putea apărea într-o instrucțiune CREATE TABLE.

```
ALTER TABLE nume_tabel
 ADD CONSTRAINT <definiţie_restricţie>;
```

```
Exemplu:
ALTER TABLE CUSTOMER_ACCOUNT
 ADD CONSTRAINT CK_CUSTOMER_DEPOSIT_AMOUNT
  CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR
         CUSTOMER_DEPOSIT_AMOUNT IS NULL);
```

- Ștergerea cheii primare a unui tabel. Dacă cheia primară este referită de restricții referențiale, trebuie mai întâi șterse restricțiile respective.

  ```
  ALTER TABLE nume_tabel DROP PRIMARY KEY;
  ```

- Redenumirea unei coloane. Dintre bazele de date studiate, numai Oracle (începând cu versiunea 8.0) acceptă această sintaxă. Totuși, Microsoft SQL Server are o procedură stocată, numită sp_rename, care vă pune la dispoziție o modalitate de a redenumi coloanele, tabelele și alte obiecte ale bazei de date.

  ```
  ALTER TABLE nume_tabel
   RENAME nume_vechi_coloană TO nume_nou_coloană;
  ```

Instrucțiunea DROP

Instrucțiunea DROP este cea mai simplă dintre instrucțiunile DDL. Sintaxa de bază este

```
DROP <tip_obiect> nume_obiect [<opţiuni_de_ştergere>]
```

- Tipul de obiect specifică tipul obiectului care urmează să fie șters, cum ar fi INDEX, TABLE sau VIEW.
- Opțiunile de ștergere sunt specifice fiecărui DBMS. În general, dacă un tabel este referit de o restricție referențială, sistemul DBMS nu vă va permite să ștergeți tabelul. Totuși, unele sisteme DBMS vă pun la dispoziție o opțiune de ștergere „în cascadă", care duc la eliminarea tuturor restricțiilor referențiale care nu permit ștergerea tabelului. Din nefericire, sintaxa este diferită de la un producător la altul – PostgreSQL și MySQL folosesc în acest scop cuvântul cheie CASCADE, în timp ce în Oracle trebuie să folosiți CASCADE CONSTRAINTS.

  Iată câteva exemple. Atenție: după ce ștergeți un obiect, *acesta dispare* – în bazele de date relaționale nu există comanda „undo", iar SQL nu vă întreabă dacă sunteți sigur că este bine ceea ce faceți.

  ```
  DROP TABLE CUSTOMER_ACCOUNT;
  DROP TABLE CUSTOMER_ACCOUNT CASCADE CONSTRAINTS; (Oracle)
  DROP TABLE CUSTOMER_ACCOUNT CASCADE; (MySQL / PostgreSQL)
  DROP INDEX IX_MOVIE_TITLE;
  ```

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Tipurile de date oferă următoarele avantaje
   a. Respectă standardele publicate
   b. Oferă un set de comportamente utile pentru utilizatorii bazei de date
   c. Asigură independența față de date
   d. Restricționează datele din coloane la caractere care au sens în context
   e. Ajută sistemul DBMS să stocheze mai eficient datele din coloane
2. Tipurile de date pentru caractere
   a. Sunt mai flexibile decât tipurile de date numerice
   b. Acceptă atât date cu lungime fixă, cât și date cu lungime variabilă
   c. Necesită întotdeauna specificarea preciziei și a scalei
   d. Determină completarea coloanelor până la lungimea maximă
   e. Pot stoca șiruri de caractere în format specific unei limbi naționale
3. Tipurile de date numerice
   a. Sunt mai flexibile decât tipurile de date pentru caractere
   b. Restricționează valorile din coloane la numere și simboluri înrudite, cum ar fi virgulele și simbolul dolar
   c. Necesită întotdeauna specificarea preciziei și a scalei
   d. Stochează valori exacte sau aproximative
   e. Sunt potrivite pentru a fi folosite în calcule
4. Tipurile numerice standard includ
   a. INTEGER
   b. NUMBER
   c. FLOAT
   d. BOOLEAN
   e. INTERVAL

5. Tipurile de date temporale standard includ
   a. DATETIME
   b. DATE
   c. TIMESTAMP
   d. TIMEZONE
   e. TIME
6. Valorile NULL
   a. Pot fi folosite pentru reprezentarea datelor care lipsesc sau nu sunt cunoscute
   b. Înseamnă același lucru ca și spațiile libere
   c. Sunt egale cu alte valori NULL
   d. Nu sunt egale cu alte valori NULL
   e. Sunt întotdeauna permise în mod prestabilit
7. Instrucțiunile DDL includ
   a. CREATE
   b. ALTER
   c. DELETE
   d. INSERT
   e. UPDATE
8. Instrucțiunea CREATE DATABASE
   a. Funcționează exact la fel în toate sistemele DBMS relaționale
   b. Specifică întotdeauna numele bazei de date
   c. Specifică întotdeauna numele proprietarului bazei de date
   d. Poate include parametri specifici producătorului
   e. Funcționează la fel cu instrucțiunea CREATE SCHEMA
9. Definiția unei coloane din instrucțiunea CREATE TABLE poate include
   a. Numele tabelului
   b. Numele coloanei

c. O restricție la nivel de tabel

d. O clauză DEFAULT

e. O clauză NULL sau NOT NULL

10. Numele unei coloane dintr-un tabel

a. Trebuie să fie specificat în instrucțiunea CREATE TABLE

b. Trebuie să fie unic în cadrul bazei de date

c. Trebuie să fie unic în cadrul tabelului

d. Poate fi folosit într-un singur index

e. Trebuie să fie specificat în instrucțiunea ALTER TABLE

11. O restricție la nivel de coloană

a. Poate referi una sau mai multe coloane

b. Poate fi inclusă într-o instrucțiune CREATE TABLE sau ALTER TABLE

c. Folosește o sintaxă identică sau aproape identică cu cea a unei restricții de același tip la nivel de tabel

d. Poate fi folosită oriunde ar putea fi folosită o restricție la nivel de tabel

e. Are o sintaxă care diferă de la un tip de restricție la altul

12. Sintaxa corectă pentru clauza DEFAULT este

a. DEFAULT (*precizie, scală*)

b. DEFAULT [NULL | NOT NULL]

c. DEFAULT (*expresie*)

d. DEFAULT (*nume_coloană*) REFERENCES *nume_tabel* (*nume_coloană*)

e. DEFAULT [UNIQUE | PRIMARY KEY]

13. Sintaxa corectă pentru o restricție NOT NULL este

a. *nume_coloană tip_de_date* IS NOT NULL

b. *nume_coloană tip_de_date* NOT NULL

c. DEFAULT [NULL | NOT NULL]

d. CREATE NOT NULL INDEX ON *nume_coloană*

e. *nume_coloană* REFERENCES NOT NULL

14. Sintaxa corectă pentru o restricție UNIQUE este
    a. [CONSTRAINT *nume_restricție*] UNIQUE (*nume_coloană*)
    b. [CONSTRAINT *nume_restricție*] UNIQUE (*nume_tabel*)
    c. DEFAULT UNIQUE (*nume_coloană*)
    d. *nume_coloană* REFERENCES UNIQUE *nume_tabel*
    e. DEFAULT [UNIQUE | PRIMARY KEY]
15. Sintaxa corectă pentru o restricție referențială este
    a. [CONSTRAINT *nume_restricție*] REFERENCES *nume_index*
    b. [CONSTRAINT *nume_restricție*] REFERENCES *nume_tabel*
    c. FOREIGN KEY *nume_coloană* REFERENCES *nume_tabel* (*nume_coloană*)
    d. REFERENCES *nume_tabel* (*nume_coloană*)
    e. *nume_coloană* REFERENCES UNIQUE *nume_tabel*
16. Instrucțiunea CREATE INDEX
    a. Poate fi folosită pentru crearea restricțiilor de unicitate și cheie primară
    b. Poate include cuvântul cheie UNIQUE
    c. Trebuie să refere două sau mai multe nume de coloane
    d. Poate include cuvintele cheie ASC sau DESC pentru orice coloană
    e. Poate specifica ordinea ascendentă sau descendentă pentru una sau mai multe coloane
17. Instrucțiunea CREATE VIEW
    a. Stochează o interogare în baza de date
    b. Poate include cuvântul cheie opțional CASCADE
    c. Poate include cuvântul cheie opțional OR REPLACE
    d. Trebuie să conțină o comandă DML validă
    e. Trebuie să conțină o instrucțiune SELECT validă

18. Utilizările valide ale instrucțiunii ALTER TABLE includ
    a. Adăugarea coloanelor
    b. Modificarea lungimii sau a preciziei coloanelor
    c. Redenumirea unui tabel
    d. Eliminarea unei chei primare
    e. Adăugarea unei chei primare
19. O instrucțiune ALTER TABLE nu poate fi folosită pentru
    a. Schimbarea tipului de date al unei coloane la un tip numeric dacă în coloana respectivă există date de alt tip
    b. Redenumirea unei coloane
    c. Schimbarea unei restricții din NULL în NOT NULL pentru o coloană care conține valori nule
    d. Eliminarea unei chei externe care referă o cheie primară
    e. Eliminarea unei chei primare dacă există chei externe care referă cheia primară
20. Instrucțiunea DROP poate fi folosită pentru a șterge
    a. O restricție referențială
    b. Un index
    c. Un tabel
    d. O coloană dintr-un tabel
    e. O vizualizare

CAPITOLUL 4

Obținerea datelor folosind limbajul de interogare a datelor (DQL)

Limbajul SQL de interogare a datelor (DQL – Data Query Language) include o singură comandă, dar una foarte importantă: SELECT. Comanda SELECT este folosită pentru a obține date din baza de date (fără să le modifice), astfel încât acestea să poate fi prelucrate de o aplicație sau afișate pentru un utilizator. Fără îndoială, este comanda SQL cea mai frecvent folosită. Mulți utilizatori folosesc termenul „instrucțiune SELECT" atunci când se referă la instrucțiunile SQL care

folosesc comanda SELECT. Rezultatul unei instrucțiuni SELECT, numit *set de rezultate*, este returnat întotdeauna sub forma unui tabel (adică rânduri și coloane). Nu uitați că SQL este un limbaj neprocedural, așa că specificați rezultatele pe care vreți să le obțineți (adică modul în care vreți să fie returnat setul de rezultate), nu și modul în care vor fi obținute acestea.

Instrucțiunea SELECT de bază

Forma elementară a instrucțiunii SELECT conține două clauze:

- **SELECT [DISTINCT]** Specifică lista de coloane care urmează să fie returnate în setul de rezultate, separate prin virgule. Puteți folosi simbolul asterisc (*) în locul listei de coloane pentru a selecta toate coloanele dintr-un tabel sau dintr-o vizualizare. Aceasta este o caracteristică utilă pentru extragerea rapidă a datelor, dar ar trebui să o evitați în instrucțiunile care vor fi refolosite, deoarece compromite independența logică a datelor – orice coloană adăugată în tabel sau în vizualizare va fi selectată automat la următoarea rulare a instrucțiunii și este posibil ca acesta să nu fie rezultatul pe care doriți să-l obțineți. Cuvântul cheie DISTINCT poate fi adăugat după cuvântul cheie SELECT pentru a elimina rândurile duplicate din rezultatele interogării.
- **FROM** Specifică lista tabelelor sau vizualizărilor din care urmează să fie selectate datele. În locul numelor reale ale tabelelor sau vizualizărilor puteți folosi *sinonime*, adică pseudonime pentru tabele sau vizualizări definite în baza de date.

De fapt, clauza FROM nu este atât de simplă cum pare. Majoritatea sistemelor DBMS relaționale pun la dispoziție funcții care returnează date de sistem, care nu sunt stocate într-un tabel sau o vizualizare, cum ar fi data și ora curente, furnizate în Oracle de funcția CURRENT_DATE și în Microsoft SQL Server de funcția getdate(). Mai mult, în Microsoft SQL Server puteți omite cuvântul cheie FROM dacă datele returnate nu provin dintr-un tabel sau o vizualizare. În majoritatea celorlalte implementări ale limbajului SQL, dacă nu în toate, clauza FROM este obligatorie. Oracle are o caracteristică neobișnuită – pune la dispoziție un tabel fictiv, numit DUAL, care poate fi folosit atunci când nici una din coloanele specificate în instrucțiunea SELECT nu provine dintr-un tabel sau o vizualizare reală.

Exemplul următor selectează coloanele MOVIE_GENRE_CODE, MPAA_RATING_CODE și MOVIE_TITLE din tabelul MOVIE. În tabelul MOVIE există 20 de filme, dar aici sunt prezentate numai primele patru.

```
SELECT MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE
  FROM MOVIE;
```

```
MOVIE_GENRE_CODE  MPAA_RATING_CODE  MOVIE_TITLE
----------------  ----------------  ----------------------------
Drama             R Mystic          River
ActAd             R The             Last Samurai
Comdy             PG-13             Something's Gotta Give
ActAd             PG-13             The Italian Job
...               ...               ...
 20 row(s) affected)
```

Observați următoarele:

- Setul de rezultate returnat a fost generat de Microsoft SQL Server. Dacă rulați aceeași instrucțiune pentru un alt DBMS, este posibil ca formatul să fie puțin diferit.
- Rândurile din setul de rezultate nu sunt returnate într-o anumită ordine. În secțiunea următoare veți învăța cum să specificați ordinea în care vreți să fie returnate rândurile.
- Au fost returnate toate rândurile din tabelul MOVIE, deoarece nu am cerut sistemului DBMS să limiteze rezultatele. Veți învăța ulterior să faceți acest lucru.
- Ultima linie din setul de rezultate este un mesaj care vă spune dacă instrucțiunea a fost rulată cu succes (mesajul este numit uneori reacție de răspuns („feedback")). Mesajul prezentat aici este afișate de Microsoft SQL Server. Alte sisteme DBMS folosesc formate diferite pentru mesaje. De exemplu, Oracle ar fi afișat mesajul „20 rows selected" în acest caz.
- Probabil ați observat că datele din coloana MOVIE_GENRE_CODE nu sunt întotdeauna clare. Este ușor să vă dați seama ce înseamnă „Drama" și „Comdy" (Comedy), dar „ActAd" (Action/Adventure) este foarte criptic. Acestea sunt coduri pentru categoriile de filme, iar coloana MOVIE_GENRE_CODE din tabelul MOVIE este cheie externă către tabelul MOVIE_GENRE. O altă soluție de proiectare ar fi fost numerotarea categoriilor, dar atunci când numărul de valori este relativ mic, eu prefer să folosesc coduri, care sunt mai ușor de reținut, fiind deseori numite valori mnemonice. În capitolul 5 veți învăța să combinați coloane din mai multe tabele într-o singură instrucțiune SELECT, printr-o operație numită *unirea* tabelelor, care în acest exemplu ne-ar fi permis să înlocuim coloana MOVIE_GENRE_CODE cu coloana MOVIE_GENRE_DESCRIPTION din tabelul MOVIE_GENRE, astfel încât rezultatul să fie mult mai prietenos pentru utilizator.
- Am indentat linia clauzei FROM astfel încât să fie frumos aliniată cu linia de deasupra ei. Acest lucru nu este necesar, dar această formatare a codului SQL este o practică bună, deoarece face codul SQL mai ușor de citit.

Pseudonime pentru numele coloanelor

Probabil ați observat în setul de rezultate din interogarea anterioară că numele coloanelor din tabel apar automat ca titluri ale coloanelor din setul de rezultate. Totuși, nu este obligatoriu să se întâmple așa, deoarece instrucțiunea SQL vă pune la dispoziție o modalitate simplă de a specifica pseudonime pentru coloane. Pseudonimele (*aliases*) specificate devin numele coloanelor din setul de rezultate. Atenție la un aspect – pseudonimele nu există decât după rularea instrucțiunii SQL, așa că nu pot fi folosite în alte părți ale instrucțiunii SQL. Pseudonimul unei coloane este specificat prin plasarea cuvântului cheie „AS" după numele coloanei în lista SELECT (cu cel puțin un spațiu înainte și după), urmat de numele pe care vreți să-l atribuiți coloanei în setul de rezultate. Multe sisteme DBMS vă permit să omiteți cuvântul cheie „AS", doar lăsând un spațiu între numele coloanei și pseudonim – motorul știe că este un pseudonim, nu un nume de coloană, deoarece numele coloanelor din lista SELECT sunt separate prin virgule. În esență, un pseudonim redenumește o coloană, dar numai în setul de rezultate. Iată cum arată instrucțiunea SQL rulată mai devreme, dar cu coloana MOVIE_GENRE_CODE redenumită GENRE și coloana MPAA_RATING_CODE redenumită RATING. Ca și în exemplul anterior, sunt prezentate aici numai primele patru rânduri:

```
SELECT MOVIE_GENRE_CODE AS GENRE,
       MPAA_RATING_CODE AS RATING, MOVIE_TITLE
  FROM MOVIE;

GENRE  RATING  MOVIE_TITLE
-----  ------  ----------------------
Drama  R       Mystic River
ActAd  R       The Last Samurai
Comdy  PG-13   Something's Gotta Give
ActAd  PG-13   The Italian Job
```

Sortarea rezultatelor

Rezultatele interogărilor sunt deseori mult mai utile dacă specificați pentru rândurile returnate o ordine care să aibă o semnificație pentru persoana sau aplicația care folosește informațiile. Nu există nici o garanție în privința ordinii în care sunt returnate rândurile din setul de rezultate *decât dacă* ordinea dorită este specificată în interogare. În SQL, acest lucru este făcut prin adăugarea în instrucțiunea SELECT a clauzei ORDER BY, cu o listă de una sau mai multe coloane care vor fi folosite pentru sortarea rândurilor în ordine ascendentă sau descendentă, în conformitate cu valorile datelor din coloane. De asemenea, țineți seama de următoarele aspecte:

- Ordinea prestabilită pentru fiecare coloană este ascendentă, dar puteți adăuga cuvântul cheie ASC după numele coloanei pentru obținerea unei ordonări ascendente sau cuvântul cheie DESC pentru obținerea unei ordonări descendente.
- Nu este obligatoriu ca numele coloanelor din lista ORDER BY să fie incluse și în lista de rezultate (adică în lista SELECT). Totuși, este mai ușor pentru persoanele care citesc rezultatele dacă faceți acest lucru.
- Motorul SQL din DBMS va găsi cea mai bună cale de ordonare a coloanelor. În general, sortarea datelor este un proces costisitor din punct de vedere al resurselor de calcul, așa că majoritatea sistemelor DBMS folosesc un index pentru accesul la rânduri în ordinea dorită, presupunând că există, și fac o sortare propriu-zisă numai ca ultimă soluție.
- *Puteți* folosi pseudonimele coloanelor în clauza ORDER BY, dar dacă faceți acest lucru forțați motorul SQL să sorteze rezultatele abia după rularea interogării. Nu poate fi folosit un eventual index pe coloana sau coloanele după care sortați datele, deoarece acel index referă numele coloanelor, nu pseudonimele. Ideea este că motorul SQL beneficiază de o flexibilitate mai mare în privința modului de ordonare a setului de rezultate dacă folosiți întotdeauna numele coloanelor în clauza ORDER BY.
- În locul coloanelor, puteți specifica în lista de ordonare poziția relativă a coloanelor. De exemplu, clauza ORDER BY 1,2 va sorta rezultatele în ordine ascendentă după primele două coloane din setul de rezultate. Numărul specificat nu are nici o legătură cu poziția coloanei în tabelul sau vizualizarea sursă. Această opțiune nu este agreată în programarea SQL formală, deoarece dacă ulterior cineva modifică interogarea, este posibil să amestece coloanele din lista SELECT, fără să-și dea seama că astfel schimbă și coloanele folosite pentru sortarea rezultatelor.

În cazul instrucțiunii SELECT folosite mai devreme, să presupunem că ar fi utilă ordonarea ascendentă a rândurilor după coloanele MPAA Rating și Movie Genre Code. Din perspectiva umană, cel mai bine este să plasați aceste coloane pe primele poziții în rezultatele interogării și să le specificați în aceeași ordinea în lista ORDER BY (cel puțin în limbile citite de la stânga la dreapta). Astfel, ordonarea rândurilor este evidentă pentru cel care citește rezultatele. Mai jos este prezentată instrucțiunea SELECT modificată, urmată de rezultatele interogării. Sunt prezentate toate cele 20 de rânduri returnate din tabelul MOVIE, astfel încât să înțelegeți cum a fost făcută ordonarea. Totuși, unele titluri au fost prescurtate, astfel încât fiecare rând să încapă pe o linie de pe pagina tipărită. Într-un client SQL obișnuit, rândurile prea lungi sunt trecute automat pe linia următoare la afișare, dar am evitat această situație aici, deoarece ar face exemplele prea greu de citit.

```
SELECT MPAA_RATING_CODE AS RATING,
       MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE
```

```
  FROM MOVIE
 ORDER BY MPAA_RATING_CODE, MOVIE_GENRE_CODE;

RATING   GENRE    MOVIE_TITLE
------   -----    ----------------------------------------------
PG-13    ActAd    The Italian Job
PG-13    ActAd    Pirates of the Caribbean: The Curse of the
PG-13    ActAd    Master and Commander: The Far Side of the
PG-13    ActAd    The Day After Tomorrow
PG-13    Comdy    50 First Dates
PG-13    Comdy    Matchstick Men
PG-13    Comdy    The School of Rock
PG-13    Comdy    Something's Gotta Give
PG-13    Drama    Big Fish
PG-13    Rmce     Two Weeks Notice
PG-13    Rmce     13 Going on 30
R        ActAd    Man on FireR ActAd Kill Bill: Vol. 1
R        ActAd    The Last SamuraiR Drama Mystic River
R        Drama    Lost in Translation
R        Drama    Road to Perdition
R        Drama    Cold MountainR Drama Monster
R        Forgn    Das Boot

(20 row(s) affected)
```

Observați că în setul de rezultate toate rândurile cu aceeași valoare MPAA Rating Code (coloana RATING) apar grupate, și în cadrul fiecărei valori MPAA Rating rândurile sunt ordonate după valoarea Movie Genre Code (coloana GENRE). Mulți profesioniști din domeniul IT (Information Technology) ar spune că setul de rezultate este ordonat după Genre *în* Rating. Doar ca să ne distrăm, să ordonăm acum rezultatele întâi ascendent după Rating și apoi *descendent* după Genre. Iată instrucțiunea SELECT modificată și setul de rezultate corespunzător (de această dată obținut dintr-o bază de date Oracle, folosind clientul SQL*Plus, astfel încât să puteți vedea diferențele în formatul setului de rezultate):

```
SELECT MPAA_RATING_CODE AS RATING,
       MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE
  FROM MOVIE
 ORDER BY MPAA_RATING_CODE ASC, MOVIE_GENRE_CODE DESC
```

```
RATING  GENRE   MOVIE_TITLE
------  -----   ----------------------------------------------
PG-13   Rmce    Two Weeks Notice
PG-13   Rmce    13 Going on 30
PG-13   Drama   Big Fish
PG-13   Comdy   Something's Gotta Give
PG-13   Comdy   50 First Dates
PG-13   Comdy   The School of Rock
PG-13   Comdy   Matchstick Men
PG-13   ActAd   The Italian Job
PG-13   ActAd   Master and Commander: The Far Side of the
PG-13   ActAd   Pirates of the Caribbean: The Curse of the
PG-13   ActAd   The Day After Tomorrow
R       Forgn   Das Boot
R       Drama   Mystic River
R       Drama   Cold Mountain
R       Drama   Road to Perdition
R       Drama   Lost in Translation
R       Drama   Monster
R       ActAd   The Last Samurai
R       ActAd   Man on Fire
R       ActAd   Kill Bill: Vol.

120 rows selected.
```

Ați observat următoarele diferențe?

- Așa cum am specificat, sortarea s-a făcut mai întâi după coloana Rating, apoi descendent după coloana Genre.
- Oracle afișează „RATIN" în loc de „RATING" ca titlu al coloanei în setul de rezultate. Pseudonimul coloanei este „RATING" dar clientul Oracle SQL*Plus prescurtează automat titlurilor coloanelor dacă sunt mai lungi decât cea mai lungă valoare din datele coloanei (mie nu mi-a plăcut niciodată această caracteristică). Vestea bună este că acest comportament poate fi modificat prin folosirea unor comenzi de formatare furnizate de SQL*Plus, dar nu vom intra în detalii aici. Vestea *și mai bună* este că noul client SQL produs de Oracle, iSQL*Plus, nu se mai comportă în acest fel. Totuși, adevărata lecție pe care trebuie să o învățați din aceste exemple este că trebuie să fiți pregătit să faceți față diferențelor de comportament dintre bazele de date și clienții SQL.
- Mesajul de terminare a execuției, afișat la sfârșitul setului de rezultate, este „20 rows selected." în loc de „(20 row(s) affected)".

Utilizarea clauzei WHERE pentru filtrarea rezultatelor

SQL folosește clauza WHERE pentru a filtra rândurile ce urmează să fie afișate. Așa cum ați văzut deja, o interogare fără o clauză WHERE returnează un set de rezultate care conține toate rândurile din tabelele sau vizualizările referite în clauza FROM. Dacă este inclusă o clauză WHERE, sunt folosite regulile algebrei booleene, numite astfel după numele logicianului George Boole, evaluând clauza WHERE pentru fiecare rând de date. În rezultatele interogării sunt afișate numai rândurile pentru care clauza WHERE este evaluată la valoarea logică „adevărat".

Așa cum veți vedea în exemplele care urmează, testele individuale ale condițiilor trebuie să fie evaluate ca „adevărat", „fals" sau „necunoscut". De fapt, starea „necunoscut" nu este o parte a algebrei booleene – provine de la folosirea valorilor nule în bazele de date relaționale (a se vedea subsecțiunea „IS NULL" din secțiunea „Operatori logici", puțin mai târziu în acest capitol).

Operatori de comparare

Operatorii de comparare sunt folosiți în clauza WHERE pentru compararea a două valori, având ca rezultat o valoare logică de „adevărat" sau „fals". Cele două valori comparate pot fi constante furnizate în clauza WHERE, valori ale unor coloane din baza de date sau combinații ale celor două. Operatorii de comparare care pot fi folosiți în clauza WHERE sunt prezentați în tabelul următor:

Operator	Descriere
=	Egal cu
<	Mai mic decât
<=	Mai mic sau egal cu
>	Mai mare decât
>=	Mai mare sau egal cu
!=	Diferit de
<>	Diferit de (standard ANSI)

Iată câteva exemple:

- Afișați toate filmele pentru care MPAA Rating are valoarea PG-13.

```
SELECT MPAA_RATING_CODE AS RATING,
       MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE = 'PG-13'
 ORDER BY MOVIE_TITLE;
```

```
RATING  MOVIE_TITLE
------  --------------------------------------------------
PG-13   13 Going on 30
PG-13   50 First Dates
PG-13   Big Fish
PG-13   Master and Commander: The Far Side of the World
PG-13   Matchstick Men
PG-13   Pirates of the Caribbean: The Curse of the Black
PG-13   Something's Gotta Give
PG-13   The Day After Tomorrow
PG-13   The Italian Job
PG-13   The School of Rock
PG-13   Two Weeks Notice11 rows selected.

11 rows selected.
```

- Afișați toate filmele pentru care MPAA Rating are altă valoare decât PG-13. Observați că în tabelul MOVIE sunt 20 de rânduri, iar DBMS a găsit 11 dintre ele care se potrivesc cu valoarea „PG-13", așa că nu este nici o surpriză că restul de 9 nu se potrivesc cu valoarea „PG-13". Totuși, dacă în coloana respectivă ar fi existat valori nule, rândurile respective nu ar fi apărut în nici unul dintre seturile de rezultate ale celor două interogări. La început vi se va părea ciudat, dar o valoare nulă nici nu este egală cu „PG-13", nici diferită de aceasta – este doar o valoare nulă. Veți afla mai multe despre valorile nule în secțiunea următoare.

```
SELECT MPAA_RATING_CODE AS RATING,
       MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE <> 'PG-13'
 ORDER BY MOVIE_TITLE;

RATING  MOVIE_TITLE
------  --------------------------------------------------
R       Cold Mountain
R       Das Boot
R       Kill Bill: Vol. 1
R       Lost in Translation
R       Man on Fire
R       Monster
R       Mystic River
R       Road to Perdition
R       The Last Samurai

9 rows selected.
```

- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (DVD Retail Price) mai mic de 19.99, în ordinea descrescătoare a prețurilor.

```
SELECT RETAIL_PRICE_DVD, MOVIE_TITLE
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD < 19.99
 ORDER BY RETAIL_PRICE_DVD DESC;

RETAIL_PRICE_DVD  MOVIE_TITLE
----------------  ----------------------------------------
           19.97  Matchstick Men
           19.96  Mystic River
           19.96  The Last Samurai
           19.94  Big Fish
           19.94  50 First Dates
           19.94  Das Boot
           14.99  Road to Perdition
           14.98  Lost in Translation
           14.97  Two Weeks Notice

9 rows selected.
```

- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (DVD Retail Price) de 19.99 sau mai mic. Observați că în rezultatele acestei interogări apare un titlu pentru care prețul este exact 19.99, care nu apărea în exemplul anterior.

```
SELECT RETAIL_PRICE_DVD, MOVIE_TITLE
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD <= 19.99
 ORDER BY RETAIL_PRICE_DVD DESC;

RETAIL_PRICE_DVD  MOVIE_TITLE
----------------  ----------------------------------------
           19.99  The Italian Job
           19.97  Matchstick Men
           19.96  Mystic River
           19.96  The Last Samurai
           19.94  Big Fish
           19.94  Das Boot
           19.94  50 First Dates
           14.99  Road to Perdition
           14.98  Lost in Translation
           14.97  Two Weeks Notice

10 rows selected.
```

- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (DVD Retail Price) mai mare de 25.00, în ordinea crescătoare a prețurilor.

```
SELECT RETAIL_PRICE_DVD, MOVIE_TITLE
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD >= 25.00
 ORDER BY RETAIL_PRICE_DVD ASC;

RETAIL_PRICE_DVD  MOVIE_TITLE
----------------  ----------------------------------------
           28.95  13 Going on 30
           29.98  Man on Fire
           29.98  The Day After Tomorrow
           29.99  Something's Gotta Give
           29.99  Kill Bill: Vol. 1
           29.99  Cold Mountain
           29.99  Monster
           29.99  The School of Rock
           29.99  Pirates of the Caribbean: The Curse of
           39.99  Master and Commander: The Far Side of

10 rows selected.
```

Operatori conjunctivi

Uneori sunt necesare condiții multiple pentru a îngusta setul de rezultate al unei interogări. Atunci când sunt folosite mai multe condiții, ele trebuie să fie combinate din punct de vedere logic în clauza WHERE, iar aceasta este sarcina *operatorilor conjunctivi*. Acești operatori sunt:

- **AND (ȘI)** Clauza WHERE este evaluată ca „adevărată" dacă *toate* condițiile conectate cu operatorul AND sunt adevărate.
- **OR (SAU)** Clauza WHERE este evaluată ca „adevărată" dacă *oricare* din condițiile conectate cu operatorul OR este adevărată.

Lucrurile devin complicate dacă operatorii AND și OR sunt combinați în aceeași clauză WHERE. Operatorul AND are prioritate mai mare și, ca urmare, este evaluat înaintea operatorilor OR. Totuși, puteți să vă faceți singur o favoare și să folosiți *întotdeauna* paranteze pentru a controla evaluarea unei clauze WHERE care combină cei doi operatori conjunctivi. Condițiile din interiorul parantezelor sunt evaluate întotdeauna primele și făcând explicit mai clar modul de evaluare pe care vreți să-l obțineți vă ajută nu numai pe dumneavoastră, ci și pe oricine ar citi ulterior interogarea.

Iată câteva exemple de folosire a operatorilor conjunctivi:

- Afișați toate filmele pentru care categoria MPAA este PG-13 *și* prețul de vânzare cu amănuntul pentru formatul DVD este 19.99 sau mai mic, în ordinea crescătoare a prețurilor.

```
SELECT MPAA_RATING_CODE AS RATING,
       RETAIL_PRICE_DVD AS PRICE,
       MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE = 'PG-13'
   AND RETAIL_PRICE_DVD <= 19.99
 ORDER BY RETAIL_PRICE_DVD;

RATING     PRICE  MOVIE_TITLE
------ --------- ----------------------------------------
PG-13      14.97  Two Weeks Notice
PG-13      19.94  Big Fish
PG-13      19.94  50 First Dates
PG-13      19.97  Matchstick Men
PG-13      19.99  The Italian Job

5 rows selected.
```

- Afișați toate filmele pentru care categoria MPAA este PG-13 *sau* prețul de vânzare cu amănuntul pentru formatul DVD este 19.99 sau mai mic, în ordinea crescătoare a prețurilor. Ar trebui să obțineți mai multe rânduri în acest set de rezultate, deoarece vor fi afișate toate filmele PG-13, indiferent de preț, și toate filmele cu prețul de 19.99 sau mai mic, indiferent de categorie.

```
SELECT MPAA_RATING_CODE AS RATING,
       RETAIL_PRICE_DVD AS PRICE,
       MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE = 'PG-13'
    OR RETAIL_PRICE_DVD <= 19.99
 ORDER BY MPAA_RATING_CODE, RETAIL_PRICE_DVD;

RATING     PRICE  MOVIE_TITLE
------ --------- ----------------------------------------
PG-13      14.97  Two Weeks Notice
R          14.98  Lost in Translation
R          14.99  Road to Perdition
PG-13      19.94  Big Fish
```

```
R           19.94  Das Boot
PG-13       19.94  50 First Dates
R           19.96  Mystic River
R           19.96  The Last Samurai
PG-13       19.97  Matchstick Men
PG-13       19.99  The Italian Job
PG-13       28.95  13 Going on 30
PG-13       29.98  The Day After Tomorrow
PG-13       29.99  Something's Gotta Give
PG-13       29.99  The School of Rock
PG-13       29.99  Pirates of the Caribbean: The Curse of
PG-13       39.99  Master and Commander: The Far Side of

16 rows selected.
```

- Afișați toate filmele pentru care categoria MPAA este PG-13 și sunt din genul dramă sau acțiune/aventură. Deși intenția din propoziția anterioară pare clară, dacă scrieți clauza WHERE exact așa, fără să folosiți paranteze, veți obține rezultate incorecte. Problema este că operatorul AND este evaluat înaintea operatorului OR, ceea ce înseamnă că veți obține toate filmele cu genul ActAd, indiferent de categoria MPAA. Iată cum arată instrucțiunea scrisă astfel, împreună cu rezultatele, așa cum *nu* intenționam să le obținem:

```
SELECT MOVIE_GENRE_CODE AS GENRE,
       MPAA_RATING_CODE AS RATING,
       MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_GENRE_CODE = 'ActAd'
    OR MOVIE_GENRE_CODE = 'Drama'
   AND MPAA_RATING_CODE = 'PG-13'
 ORDER BY MOVIE_GENRE_CODE, MPAA_RATING_CODE;

GENRE  RATING   MOVIE_TITLE
-----  ------   ----------------------------------------
ActAd  PG-13    The Italian Job
ActAd  PG-13    Pirates of the Caribbean: The Curse of
ActAd  PG-13    The Day After Tomorrow
ActAd  PG-13    Master and Commander: The Far Side of
ActAd  R        The Last Samurai
ActAd  R        Man on Fire
ActAd  R        Kill Bill: Vol. 1
Drama  PG-13    Big Fish

8 rows selected.
```

- Să adăugăm acum parantezele necesare, astfel încât să obținem filmele cu categoria PG-13 și genul acțiune/aventură sau dramă. Iată instrucțiunea corectă și rezultatele pe care intenționam să le obținem:

```
SELECT MOVIE_GENRE_CODE AS GENRE,
       MPAA_RATING_CODE AS RATING,
       MOVIE_TITLE
  FROM MOVIE
 WHERE (MOVIE_GENRE_CODE = 'ActAd'
    OR MOVIE_GENRE_CODE = 'Drama')
   AND MPAA_RATING_CODE = 'PG-13'
 ORDER BY MOVIE_GENRE_CODE, MPAA_RATING_CODE;

GENRE  RATING   MOVIE_TITLE
-----  ------   ----------------------------------------
ActAd  PG-13    The Italian Job
ActAd  PG-13    Pirates of the Caribbean: The Curse of
ActAd  PG-13    Master and Commander: The Far Side of
ActAd  PG-13    The Day After Tomorrow
Drama  PG-13    Big Fish

5 rows selected.
```

Operatori logici

Operatorii logici folosesc cuvinte cheie în locul simbolurilor la formarea expresiilor de comparare. Operatorii logici disponibili în majoritatea implementărilor SQL sunt prezentați în secțiunile care urmează. La oricare dintre acești operatori poate fi adăugat cuvântul cheie NOT, pentru a inversa valoarea logică a comparației.

IS NULL

Operatorul IS NULL este folosit pentru a determina dacă o valoare este nulă. Este important să rețineți că valorile nule din baza de date nu sunt egale cu nimic altceva, nici chiar cu alte valori nule. Din această cauză, o condiție precum „= NULL" este întotdeauna incorectă. Niciodată nimic nu este egal cu o valoare nulă. Iată câteva exemple:

- Găsiți toate conturile de clienți active, adică toate conturile pentru care coloana DATE_TERMINATED conține o valoare nulă:

```
SELECT CUSTOMER_ACCOUNT_ID
  FROM CUSTOMER_ACCOUNT
 WHERE DATE_TERMINATED IS NULL;
```

```
CUSTOMER_ACCOUNT_ID
-------------------
                  1
                  2
                  3
                  4
                  5
                  7
                  8
                  9

8 rows selected.
```

- Găsiți toate conturile de clienți inactive, adică toate conturile pentru care coloana DATE_TERMINATED conține o altă valoare decât NULL:

```
SELECT CUSTOMER_ACCOUNT_ID
  FROM CUSTOMER_ACCOUNT
 WHERE DATE_TERMINATED IS NOT NULL;

CUSTOMER_ACCOUNT_ID
-------------------
                  6

1 row selected.
```

BETWEEN

Operatorul BETWEEN este folosit pentru a determina dacă o valoare se încadrează într-un interval specificat. Intervalul este specificat folosind o valoare minimă și o valoare maximă, fiind un interval *inclusiv*, ceea ce înseamnă că include și valorile specificate. Aceasta este o prescurtare elegantă, ușor de citit și de înțeles, pentru specificare unei condiții de interval. De exemplu, condiția „WHERE MOVIE_ID BETWEEN 7 AND 9" este aceeași cu condiția „WHERE MOVIE_ID >= 7 AND MOVIE_ID <= 9. Iată câteva exemple:

- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (coloana RETAIL_PRICE_DVD) între 14.99 și 19.99, ordonate crescător după preț. Observați că în setul de rezultate sunt incluse și filmele pentru care prețul este exact 14.99 sau 19.99.

```
SELECT MOVIE_TITLE, RETAIL_PRICE_DVD
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD BETWEEN 14.99 AND 19.99
 ORDER BY RETAIL_PRICE_DVD;
```

```
MOVIE_TITLE          RETAIL_PRICE_DVD
-------------------  ----------------
Road to Perdition                14.99
Big Fish                         19.94
Das Boot                         19.94
50 First Dates                   19.94
Mystic River                     19.96
The Last Samurai                 19.96
Matchstick Men                   19.97
The Italian Job                  19.99

8 rows selected.
```

- Afișați toate filmele pentru care prețul de vânzare cu amănuntul pentru formatul DVD (coloana RETAIL_PRICE_DVD) *nu* este în intervalul 14.99-19.99, ordonate crescător după preț. Observați că în setul de rezultate nu sunt incluse filmele pentru care prețul este exact 14.99 sau 19.99, fiind selectate numai filmele pentru care prețul este mai mic de 14.99 sau mai mare de 19.99.

```
SELECT MOVIE_TITLE, RETAIL_PRICE_DVD
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD NOT BETWEEN 14.99 AND 19.99
 ORDER BY RETAIL_PRICE_DVD;
```

```
MOVIE_TITLE          RETAIL_PRICE_DVD
-------------------  ----------------
Two Weeks Notice                 14.97
Lost in Translation              14.98
13 Going on 30                   28.95
Man on Fire                      29.98
The Day After Tomorr             29.98
Something's Gotta Gi             29.99
Cold Mountain                    29.99
Monster                          29.99
The School of Rock               29.99
Kill Bill: Vol. 1                29.99
Pirates of the Carib             29.99
Master and Commander             39.99

12 rows selected.
```

- Afișați toate conturile de clienți deschise în luna ianuarie 2005. Observați că operatorii logici funcționează pentru coloanele care conțin date calendaristice, la fel ca și pentru date numerice sau caractere:

```
SELECT CUSTOMER_ACCOUNT_ID, DATE_ENROLLED
  FROM CUSTOMER_ACCOUNT
 WHERE DATE_ENROLLED BETWEEN '2005/01/01'
                         AND '2005/01/31';

CUSTOMER_ACCOUNT_ID   DATE_ENROLLED
-------------------   -------------
                  1   2005-01-01
                  2   2005-01-18

2 rows selected.
```

NOTĂ: În Oracle trebuie să schimbați formatul prestabilit al datelor calendaristice pentru ca acest exemplu să funcționeze corect. Instrucțiunea de mai jos schimbă formatul datelor pentru sesiunea curentă a bazei de date, astfel încât să fie identic cu formatul folosit în exemplul anterior:

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD'
```

LIKE

Operatorul LIKE este folosit pentru a compara o valoare de tip caracter cu un tipar*, returnând valoarea logică „adevărat" dacă valoarea de tip caracter se încadrează în tipar și „fals" în caz contrar. Pentru definirea tiparului pot fi folosite două caractere de înlocuire:

- **Liniuța de subliniere (_)** Caracterul liniuță de subliniere poate fi folosit drept caracter de înlocuire pozițional, ceea ce înseamnă că se potrivește cu orice caracter aflat pe poziția respectivă în șirul de caractere evaluat.
- **Procent (%)** Simbolul procent (%) poate fi folosit drept caracter de înlocuire nepozițional, ceea ce înseamnă că se potrivește cu orice număr de caractere, indiferent de lungime.

Rețineți că Microsoft Access oferă o caracteristică similară, dar pentru caracterul de înlocuire pozițional este folosit semnul de întrebare (?), iar pentru caracterul de înlocuire nepozițional este folosit asteriscul (*) – aceste caractere corespund convențiilor folosite în DOS și Visual Basic.

---

* *pattern* în limba engleză. Numit și *șablon* sau *model*. (nota trad.)

Iată câteva exemple de tipare, pentru a vă ajuta să vizualizați utilizarea tiparelor care conțin caractere de înlocuire:

Tipar	Interpretare
%Now	Se potrivește cu orice șir de caractere care se termină cu „Now".
Now%	Se potrivește cu orice șir de caractere care începe cu „Now".
%Now%	Se potrivește cu orice șir de caractere care conține „Now" (la început, la sfârșit sau în mijloc).
N_w	Se potrivește cu orice șir de caractere format din exact trei caractere, care începe cu „N" și se termină cu „w".
%N_w%	Se potrivește cu orice șir de caractere care conține litera „N", urmată de orice alt caracter, urmat de litera „w" (la începutul, la sfârșitul sau undeva în mijlocul șirului de caractere).

Rețineți că datele din bazele de date relaționale fac *întotdeauna* diferențierea literelor mari de cele mici. O literă mică din date *nu* se potrivește cu o literă mare din tiparul unei clauze LIKE, și invers. Totuși, puteți folosi funcțiile UPPER și LOWER, prezentate mai târziu în acest capitol, pentru a schimba tipul caracterelor din datele provenite din baza de date, astfel încât să simplificați puțin compararea datelor cu tipuri de litere amestecate.

Iată un exemplu de utilizare a operatorului LIKE:

- Afișați toate titlurile de filme care conțin șirul de caractere „on":

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%on%';

MOVIE_TITLE
--------------------
13 Going on 30
Lost in Translation
Man on Fire
Monster
Road to Perdition

5 rows selected.
```

Probabil nu sunt exact rezultatele pe care doreați să le obțineți. Dacă intenționați să găsiți titlurile care conțin cuvântul „on", nu literele „on" din alte cuvinte, ar fi trebuit să includeți în tipar și spațiile necesare, ca în exemplul următor:

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_TITLE LIKE '% on %';

MOVIE_TITLE
--------------------
13 Going on 30
Man on Fire

2 rows selected.
```

IN

Operatorul IN este folosit pentru a determina dacă o valoare face parte dintr-o listă de valori. Lista poate fi specificată ca valori literale, folosind o listă de valori separate prin virgule și încadrată între paranteze, sau poate fi selectată din baza de date folosind o *subselecție* (o *subinterogare*), care este o interogare în cadrul unei alte interogări. Subinterogările sunt prezentate în detaliu în capitolul 5. Iată câteva exemple de utilizare a operatorului IN:

- Afișați toate filmele pentru care MOVIE_GENRE_CODE este Drama, Forgn sau Rmce. Evident, această interogare ar putea fi scrisă folosind trei condiții de egalitate, separate prin operatorul logic OR, dar este mult mai simplu să o scriem cu ajutorul operatorului IN:

```
SELECT MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_GENRE_CODE IN ('Drama','Forgn','Rmce')
 ORDER BY MOVIE_GENRE_CODE, MOVIE_TITLE;

GENRE   MOVIE_TITLE
-----   ------------------------
Drama   Big Fish
Drama   Cold Mountain
Drama   Lost in Translation
Drama   Monster
Drama   Mystic River
Drama   Road to Perdition
Forgn   Das Boot
Rmce    13 Going on 30
Rmce    Two Weeks Notice

9 rows selected.
```

- Afișați toate filmele pentru care descrierea genului conține cuvântul „and". Aveți nevoie de o subinterogare prin care să găsiți toate valorile MOVIE_GENRE_CODE care conțin cuvântul „and" în descriere. Operatorul IN este apoi folosit pentru a găsi filmele care au unul dintre codurile selectate de subinterogare. Vom discuta mai mult despre subinterogări în capitolul 5. Pentru moment, rețineți că este executată mai întâi interogarea internă (cea din paranteze), iar rezultatele returnate de interogarea internă sunt folosite ca listă de valori pentru operatorul IN din interogarea externă. Observați că deși există mai multe genuri cu descrieri care conțin cuvântul „and", numai pentru unul dintre aceste genuri (Action and Adventure - „ActAd") există în acest moment filme în magazinul de produse video. Desigur, aceasta nu este o problemă, deoarece interogarea poate fi rulată ulterior, când este posibil ca inventarul magazinului să conțină și celelalte genuri.

```
SELECT MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_GENRE_CODE IN
   (SELECT MOVIE_GENRE_CODE
      FROM MOVIE_GENRE
     WHERE MOVIE_GENRE_DESCRIPTION LIKE '% and %')
 ORDER BY MOVIE_GENRE_CODE, MOVIE_TITLE;

GENRE  MOVIE_TITLE
-----  ----------------------------------------
ActAd  Kill Bill: Vol. 1
ActAd  Man on Fire
ActAd  Master and Commander: The Far Side of the
ActAd  Pirates of the Caribbean: The Curse of the
ActAd  The Day After Tomorrow
ActAd  The Italian Job
ActAd  The Last Samurai

7 rows selected.
```

EXISTS

Operatorul EXISTS este folosit pentru a determina dacă o subinterogare conține înregistrări. Dacă în setul de rezultate al subinterogării nu există nici un rând, operatorul returnează valoarea logică „false"; dacă setul de rezultate conține cel puțin un rând, valoarea logică devine „adevărat". Repet, vom discuta mai mult despre subinterogări în capitolul 5, dar prezentăm aici un exemplu de utilizare a operatorului EXISTS, pentru a înțelege cum funcționează:

- Proprietarul magazinului a auzit că filmul *The Last Samurai* se închiriază bine atât în format DVD, cât și-n format VHS și vrea să se asigure că în inventarul magazinului există o copie VHS. Tabelul MOVIE_COPY conține un rând pentru fiecare copie a unui film din inventarul magazinului, așa că puteți folosi o subinterogare pentru a afla dacă există copii VHS ale filmului în inventar. De cele mai multe ori, operatorul EXISTS este folosit în conjuncție cu o formă mai complexă de subinterogare, numită *subinterogare corelată (correlated subquery)*, în care valorile datelor din interogarea externă (MOVIE_ID, în acest caz) sunt comparate cu rândurile din interogarea internă. Nu vă faceți prea multe griji dacă vi se pare că sintaxa acestui exemplu este confuză, deoarece vom reveni asupra ei în capitolul 5. pentru moment, este suficient să știți că subinterogarea va returna un rând dacă există o copie VHS în stoc și nu va returna nici un rând dacă nu există o astfel de copie. Aceeași interogare ar putea fi scrisă și folosind operatorul IN sau folosind o *uniune*, care leagă rândurile din două tabele. Despre uniuni vom discuta tot în capitolul 5.

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE m
 WHERE MOVIE_TITLE = 'The Last Samurai'
   AND EXISTS
  (SELECT MOVIE_ID
     FROM MOVIE_COPY c
    WHERE m.MOVIE_ID = c.MOVIE_ID);

MOVIE_ID   MOVIE_TITLE
--------   -----------
       2   The Last Samurai

1 row selected.
```

- Dacă inversați logica, folosind operatorul NOT EXISTS, puteți afișa titlul filmului numai dacă *nu* există o copie VHS în inventar.

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE m
 WHERE MOVIE_TITLE = 'The Last Samurai'
   AND NOT EXISTS
  (SELECT MOVIE_ID
     FROM MOVIE_COPY c
    WHERE m.MOVIE_ID = c.MOVIE_ID);

no rows selected
```

Operatori aritmetici

În SQL, operatorii aritmetici sunt folosiți pentru efectuarea calculelor matematice – la fel ca și-n formulele dintr-o foaie de calcul tabelar sau într-un limbaj de programare, precum Java sau C. Cei patru operatori aritmetici din SQL sunt:

Operator	Descriere
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire

Ca și în cazul operatorilor conjunctivi, dacă amestecați operatorii aritmetici în aceeași instrucțiune SQL fără să folosiți paranteze, ordinea în care sunt evaluate operațiile este determinată de prioritatea predefinită. Din fericire, prioritatea operatorilor din SQL este cea pe care o folosim în operațiile matematice obișnuite. În școala primară, am învățat că operațiile sunt efectuate în ordinea paranteze, înmulțire, împărțire, adunare și scădere. Totuși, *întotdeauna* este mai bine să folosiți paranteze, astfel încât să nu fie nevoie să țineți minte ordinea de efectuare operațiilor atunci când citiți o instrucțiune SQL, dumneavoastră sau altcineva. Iată câteva exemple de utilizare a operatorilor aritmetici:

- Cât v-ar costa să cumpărați copiile VHS și DVD ale filmului *The Last Samurai*? Observați că folosesc un pseudonim pentru numele coloanei din setul de rezultate. Dacă nu faceți acest lucru, motorul SQL este forțat să genereze un nume pentru coloana respectivă din setul de rezultate, deoarece valoarea coloanei este produsă de o funcție, nu de o coloană din baza de date. În general, numele generate nu sunt ușor de citit, așa că vă recomand să folosiți întotdeauna pseudonime pentru coloanele calculate.

```
SELECT RETAIL_PRICE_VHS + RETAIL_PRICE_DVD AS COST
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai';

     COST
---------
    35.91
```

- Cât v-ar costa aceeași achiziție dacă ați avea un bon valoric de 5$? Deși, din punct de vedere tehnic, nu sunt necesare, am adăugat paranteze pentru a face instrucțiunea mai ușor de citit.

```
SELECT (RETAIL_PRICE_VHS + RETAIL_PRICE_DVD) - 5 AS COST
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai';

     COST
---------
    30.91
```

- Dacă taxele sunt de 8.25% (0.0825), cât reprezintă taxele de vânzare din costul achiziției anterioare? Presupunem că taxele sunt calculate pentru întregul preț (adică înainte de scăderea bonurilor valorice). În acest exemplu, parantezele sunt esențiale pentru obținerea rezultatului corect – trebuie să vă asigurați că adunarea este făcută *înaintea* înmulțirii. De asemenea, s-ar putea să observați că am omis spațiile dinaintea și de după operatorul de adunare (+). Deși instrucțiunea este mai dificil de citit în acest fel, rețineți că în SQL nu este obligatoriu ca operatorii aritmetici să aibă spații înainte și după.

```
SELECT (RETAIL_PRICE_VHS+RETAIL_PRICE_DVD) * 0.0825 AS TAX
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai';

      TAX
---------
   2.8728
```

  Probabil ați observat că rezultatul are mai mult de două zecimale, ceea ce nu corespunde cu precizia pentru moneda americană (dolari și cenți). Totuși, încă nu ați învățat să faceți rotunjirea numerelor. Rotunjirile pot fi făcute cu ajutorul unei *funcții* SQL, despre care vom discuta în secțiunea „Funcții SQL elementare", așa că în curând veți ști să faceți și rotunjiri.

- Care este costul mediu pentru o copie a filmului *The Last Samurai*? Veți afla acest lucru adunând prețul unei copii DVD cu cel al unei copii VHS și împărțind rezultatul la 2. Și în acest caz, parantezele sunt esențiale pentru obținerea unui rezultat corect. Probabil veți observa că și acest rezultat poate beneficia de pe urma rotunjirii.

```
SELECT (RETAIL_PRICE_VHS + RETAIL_PRICE_DVD) / 2 AS AVG_COST
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai'

 AVG_COST
---------
   17.955
```

Funcții SQL elementare

O *funcție* este un tip special de program, care returnează o singură valoare de fiecare dată când este apelată. Termenul provine de la conceptul matematic al unei funcții. În SQL, funcțiile necesită întotdeauna specificarea unei expresii, care deseori include numele unei coloane. Cel mai des, funcțiile sunt folosite în lista de coloane a unei instrucțiuni SELECT, sunt apelate pentru fiecare rând prelucrat de interogare și, ca urmare, returnează o singură valoare pentru fiecare rând din setul de rezultate. Uneori este folosit termenul *funcție de coloană*, pentru a indica faptul că funcție este aplicată unei coloane dintr-un tabel sau o vizualizare. Un număr de funcții sunt furnizate de producătorul DBMS și, în cele mai multe implementări SQL, puteți scrie propriile funcții, folosind un limbaj special livrat împreună cu sistemul DBMS, cum ar fi PL/SQL pentru Oracle sau Transact SQL pentru Microsoft SQL Server și Sybase Adaptive Server.

Funcțiile pot fi clasificate în multe moduri, dar majoritatea specialiștilor le împart după ceea ce fac. În continuare sunt descrise funcțiile folosite cel mai frecvent, dar, ca întotdeauna, consultați documentația sistemului DBMS – probabil acesta vă pune la dispoziția mult mai multe funcții decât veți întâlni aici. În această carte am folosit categoriile caracter, matematice, de conversie și de agregare, fiecare categorie având propriile secțiuni.

Funcții pentru caractere

Funcțiile pentru caractere sunt numite astfel deoarece manipulează date de tip text.

Concatenarea șirurilor de caractere

Funcția de *concatenare a șirurilor de caractere* reunește mai multe șiruri de caractere pentru a forma o singură valoare în rezultatele interogării. Funcția standard de concatenare a șirurilor de caractere din SQL este apelată cu două bare verticale (||), dar există și excepții, cum ar fi Microsoft SQL Server, care folosește semnul plus (+) pentru concatenarea șirurilor de caractere. Iată câteva exemple de concatenare a șirurilor de caractere:

- Magazinul de produse video vrea să trimită fiecărui client o scrisoare care începe cu formula „Valuable Customer", plus prenumele și numele persoanei. Numele sunt stocate în tabelul PERSON. Remarcați că puteți combina cum doriți șirurile literale (inclusiv spații) și datele din coloane. Iată soluția acestei cerințe în Oracle:

```
SELECT 'Valuable Customer ' || PERSON_GIVEN_NAME ||
       ' ' || PERSON_FAMILY_NAME AS CUSTOMER_SALUTATION
  FROM PERSON;
```

```
CUSTOMER_SALUTATION
-----------------------------------
Valuable Customer Austin Alexander
Valuable Customer Tin Chung
Valuable Customer Cassandra Alvarado
Valuable Customer Raul Alvarado
Valuable Customer Klaus Schmidt
Valuable Customer Katarina Schmidt
Valuable Customer Karl Schmidt
Valuable Customer Toshiro Yamada
Valuable Customer Beverly Baker
Valuable Customer Gerald Bernstein
Valuable Customer Rose Bernstein
Valuable Customer Steven Bernstein
Valuable Customer Linda Campos
Valuable Customer Jorge Jimenez
Valuable Customer Liyi Huang

15 rows selected.
```

- Iată cum arată aceeași soluție, modificată pentru a funcționa în Microsoft SQL Server (setul de rezultate a fost omis, deoarece este, practic, identic cu cel dinainte):

```
SELECT 'Valuable Customer ' + PERSON_GIVEN_NAME +
       ' ' + PERSON_FAMILY_NAME AS CUSTOMER_SALUTATION
  FROM PERSON;
```

UPPER

Funcția UPPER transformă literele dintr-un șir de caractere în litere mari. Numerele și caracterele speciale sunt lăsate ca atare. Iată câteva exemple:

- Afișați comediile (MOVIE_GENRE_CODE = 'Comdy') scriind titlurile cu majuscule. Remarcați că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.

```
SELECT UPPER(MOVIE_TITLE) AS MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_GENRE_CODE = 'Comdy';


MOVIE_TITLE
------------------------
SOMETHING'S GOTTA GIVE
```

```
50 FIRST DATES
MATCHSTICK MEN
THE SCHOOL OF ROCK

4 rows selected.
```

- De asemenea, funcția UPPER este deseori folosită în condițiile WHERE. Să presupunem că nu vă amintiți dacă valorile MOVIE_GENRE_CODE au fost stocate cu litere mari, litere mici sau combinații ale acestora. Dacă în condiția WHERE transformați valorile în litere mari, puteți obține rezultatele corecte indiferent de modul de stocare.

```
SELECT UPPER(MOVIE_TITLE) AS MOVIE_TITLE
  FROM MOVIE
 WHERE UPPER(MOVIE_GENRE_CODE) = 'COMDY';

MOVIE_TITLE
----------------------
50 FIRST DATES
MATCHSTICK MEN
SOMETHING'S GOTTA GIVE
THE SCHOOL OF ROCK

4 rows selected.
```

Atenție la folosirea funcțiilor SQL în condițiile WHERE. În cele mai multe situații, pentru o coloană căreia îi este aplicată o funcție nu poate fi folosită indexarea. Ca urmare, în cazul tabelelor mari, utilizarea funcțiilor în condițiile WHERE poate duce la probleme de performanță cu adevărat memorabile. Ați observat că ordinea rândurilor din setul de rezultate s-a schimbat atunci când a fost folosită funcție UPPER în clauza WHERE? Pentru interogările anterioare am folosit Oracle, așa că rezultatele obținute în alte sisteme DBMS pot fi diferite. Dar de ce s-a întâmplat acest lucru? În primul rând, nu am folosit o clauză ORDER BY, așa că sistemul DBMS nu a fost forțat să returneze rândurile într-o anumită ordine. În al doilea rând, în primul exemplu sistemul DBMS a folosit un index pentru a găsi rândurile, în timp ce în al doilea exemplu a parcurs tot tabelul; evident, indexul nu are aceeași ordine ca și rândurile din tabel. Ce lecție învățați de aici? Că trebuie să specificați *întotdeauna* clauza ORDER BY dacă ordinea în care apar rezultatele este importantă.

LOWER

Funcția LOWER este inversa funcției UPPER – transformă literele dintr-un șir de caractere în litere mici. Iată câteva exemple de utilizare a funcției LOWER:

- Afișați comediile (MOVIE_GENRE_CODE = 'Comdy') scriind titlurile cu minuscule. Remarcați că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.

```
SELECT LOWER(MOVIE_TITLE) AS MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_GENRE_CODE = 'Comdy';

MOVIE_TITLE
------------------------
something's gotta give
50 first dates
matchstick men
the school of rock

4 rows selected.
```

- Ca și în cazul funcției UPPER, puteți folosi funcția LOWER într-o clauză WHERE, atunci când nu știți sigur ce tip de litere conține textul pe care vreți să-l comparați. Afișați toate filmele care au în titlu cuvântul „of", indiferent dacă este scris cu litere mari sau mici. Observați folosirea combinată a operatorului LIKE și a funcției LOWER. De asemenea, observați că trebuie să țineți seama de situațiile speciale, în care „of" este primul sau ultimul cuvânt din titlu (adică atunci când nu are un spațiu înainte sau după). În exemplul nostru, aceste cazuri nu există în baza de date, dar ar putea exista, și nu puteți folosi expresia LIKE '%OF%', deoarece astfel veți selecta și titlurile care conțin cuvinte precum „often". Morala este că trebuie să știți câte ceva despre ceea ce *ar putea* fi în baza de date pentru a scrie instrucțiuni SQL care produc *întotdeauna* rezultatele așteptate.

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE LOWER(MOVIE_TITLE) LIKE '% of %'
    OR LOWER(MOVIE_TITLE) LIKE 'of%'
    OR LOWER(MOVIE_TITLE) LIKE '% of';

MOVIE_TITLE
------------------------------------------------------
Master and Commander: The Far Side of the World
Pirates of the Caribbean: The Curse of the Black Pearl
The School of Rock

3 rows selected.
```

SUBSTR

Funcția SUBSTR apare în majoritatea implementărilor SQL, dar uneori are un nume puțin diferit. De exemplu, funcția se numește SUBSTRING în Microsoft SQL Server, Sybase Adaptive Server și MySQL, dar SUBSTR în Oracle și DB2. Funcția returnează o porțiune a șirului de caractere, în funcție de parametrii furnizați, care specifică numele coloanei, poziția de început a subșirului în datele coloanei și lungimea subșirului returnat (numărul de caractere). Deși este o utilizare mai puțin obișnuită, funcția SUBSTR acceptă și un șir de caractere literal în locul numelui unei coloane. Iată forma generală a funcției, urmată de un exemplu:

```
SUBSTR (numele_coloanei, poziţia_de_început, lungimea_subşirului)
```

- În tabelul PERSON, unele persoane au al doilea nume în întregime, alții au numai inițiala. Afișați numele complet al persoanelor al căror nume de familie începe cu litera „B", sub forma unui singur șir de caractere care conține prenumele, inițiala și numele de familie.

Iată soluția, pentru Oracle:

```
SELECT PERSON_GIVEN_NAME || ' ' ||
       SUBSTR(PERSON_MIDDLE_NAME, 1, 1) || '. ' ||
       PERSON_FAMILY_NAME AS FULL_NAME
  FROM PERSON
 WHERE SUBSTR(PERSON_FAMILY_NAME, 1, 1) = 'B'

FULL_NAME
--------------------
Beverly V. Baker
Gerald M. Bernstein
Rose B. Bernstein
Steven R. Bernstein

4 rows selected.
```

Observați folosirea funcției SUBSTR în clauza WHERE pentru a elimina din rezultate persoanele al căror nume de familie nu începe cu litera „B". Ar trebui să vă puteți deja gândi la alte moduri de a face acest lucru, prin folosirea operatorului LIKE.

Iată și versiunea pentru Microsoft SQL Server a exemplului anterior:

```
SELECT PERSON_GIVEN_NAME + ' ' +
       SUBSTRING(PERSON_MIDDLE_NAME, 1, 1) + '. ' +
       PERSON_FAMILY_NAME AS FULL_NAME
  FROM PERSON
 WHERE SUBSTRING(PERSON_FAMILY_NAME, 1, 1) = 'B'
```

```
FULL_NAME
--------------------
Beverly V. Baker
Gerald M. Bernstein
Rose B. Bernstein
Steven R. Bernstein

 4 row(s) affected)
```

LENGTH

Funcția LENGTH returnează lungimea unui șir de caractere. Rețineți că implementările Microsoft SQL Server și Sybase Adaptive Server folosesc numele LEN pentru versiunea proprie a acestei funcții. Iată câteva exemple:

- Afișați lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiți o bază de date Oracle, DB2 sau MySQL.

```
SELECT MOVIE_TITLE, LENGTH(MOVIE_TITLE) AS LENGTH
  FROM MOVIE
 WHERE MOVIE_ID = 1;

MOVIE_TITLE                LENGTH
------------------   ------------
Mystic River                 12
```

- Afișați lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiți o bază de date Microsoft SQL Server.

```
SELECT MOVIE_TITLE, LEN(MOVIE_TITLE) AS LENGTH
  FROM MOVIE
 WHERE LEN(MOVIE_TITLE) < 10;

MOVIE_TITLE             LENGTH
---------------   ------------
Big Fish                     8
Das Boot                     8
Monster                      7
```

Funcții matematice

Funcțiile matematice manipulează valori numerice, în conformitate cu regulile matematicii. Funcția ROUND este prezentată în detaliu, inclusiv cu un exemplu, apoi urmează un tabel cu funcțiile matematice existente în majoritatea implementărilor SQL. Ca întotdeauna,

consultați documentația sistemului DBMS pentru o listă specifică a funcțiilor matematice acceptate.

ROUND

Funcția ROUND rotunjește o valoare la un număr specificat de zecimale. Valoarea numerică este furnizată prin primul parametru, iar numărul de zecimale prin cel de-al doilea. În continuare este prezentat formatul general al funcției ROUND, urmat de un exemplu, adaptat dintr-un exemplu folosit anterior în acest capitol.

```
ROUND (expresie_numerică, număr_de_poziţii_zecimale)
```

- Care este costul mediu al unei copii a filmului *The Last Samurai*, rotunjit la două zecimale?

```
SELECT ROUND((RETAIL_PRICE_VHS + RETAIL_PRICE_DVD) / 2, 2)
       AS AVG_COST
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai';

   AVG_COST
-----------
       17.96
```

Alte funcții matematice

Tabelul care urmează prezintă funcțiile matematice cel mai des întâlnite. Pentru toate, sintaxa generală este aceeași:

```
NUME_FUNCŢIE (expresie)
```

Funcție	Descriere
ABS	Valoarea absolută a unui număr dat
COS	Cosinusul trigonometric al unui unghi specificat în radiani
EXP	Valoarea exponențială a unui număr dat
POWER	Ridică un număr la o putere (numărul și puterea sunt furnizate ca parametri)
SIN	Sinusul trigonometric al unui unghi specificat în radiani
TAN	Tangenta trigonometrică a unui unghi specificat în radiani

Funcții de conversie

Funcțiile de conversie transformă date dintr-un tip de date în altul.

CAST

Funcția CAST transformă date dintr-un tip de date în altul. Rețineți că această funcție nu este implementată de DB2. Iată sintaxa generală a funcției CAST, urmată de un exemplu:

```
CAST (expresie AS tip_de_date)
```

- Afișați prețul pentru formatul DVD al filmului *The Last Samurai*, cu un simbol dolar în fața sumei. Valoarea numerică trebuie să fie convertită într-un șir de caractere pentru a putea fi concatenată cu o valoare literală conținând simbolul dolar. Rețineți că pentru Microsoft SQL Server operatorul de concatenare din instrucțiunea următoare trebuie să fie schimbat din „||" în „+".

```
SELECT '$' || CAST(RETAIL_PRICE_DVD AS VARCHAR(6)) AS PRICE
  FROM MOVIE
 WHERE MOVIE_TITLE = 'The Last Samurai';

PRICE
-------
$19.96
```

CONVERT TO

Multe implementări DBMS pun la dispoziție o funcție CONVERT sau CONVERT TO. Totuși, în general este recomandată folosirea funcției CAST, deoarece este implementată într-un mod mai unitar de către diferiți producători.

Funcții de agregare și gruparea rândurilor

O *funcție de agregare (aggregate functions)* este o funcție care combină mai multe rânduri de date într-un singur rând. Tabelul următor prezintă funcțiile de agregare acceptate în majoritatea implementărilor SQL:

Funcție	Descriere
AVG	Calculează valoarea medie pentru o coloană sau o expresie.
COUNT	Numără valorile dintr-o coloană. Puteți folosi cuvântul cheie distinct pentru a număra valorile unice, în locul tuturor valorilor (rândurilor) dintr-o coloană.
MAX	Găsește valoarea maximă dintr-o coloană.
MIN	Găsește valoarea minimă dintr-o coloană.
SUM	Însumează valorile dintr-o coloană.

Iată câteva exemple:

- Care este prețul mediu al unui DVD? Observați că funcțiile ROUND și AVG sunt imbricate, astfel încât să obțineți rezultatul în dolari și cenți.

```
SELECT ROUND(AVG(RETAIL_PRICE_DVD),2) AS AVG_PRICE
  FROM MOVIE;

AVG_PRICE
---------
    24.67
```

- Câte filme există în tabelul MOVIE?

```
SELECT COUNT(*) AS NUM_MOVIES
  FROM MOVIE;

NUM_MOVIES
----------
        20
```

- Câte genuri diferite de filme sunt reprezentate în tabelul MOVIE? Observați folosirea cuvântului cheie DISTINCT, astfel încât sistemul DBMS să numere numai valorile MOVIE_GENRE_CODE unice.

```
SELECT COUNT(DISTINCT(MOVIE_GENRE_CODE)) AS NUM_GENRES
  FROM MOVIE

 NUM_GENRES
-----------
          5
```

- Care sunt lungimea minimă și maximă a titlurilor filmelor? Observați imbricarea funcției LENGTH cu funcțiile MIN și MAX. Această versiune funcționează în Oracle, DB2 și MySQL (pentru Microsoft SQL Server și Sybase înlocuiți LENGTH cu LEN).

```
SELECT MIN(LENGTH(MOVIE_TITLE)) AS MIN_LENGTH,
       MAX(LENGTH(MOVIE_TITLE)) AS MAX_LENGTH
  FROM MOVIE;

MIN_LENGTH  MAX_LENGTH
----------  ----------
         7          54
```

Clauza GROUP BY

Așa cum ați observat, dacă folosiți o funcție de agregare, ca atare, într-o interogare, obțineți ca rezultat un singur rând din întreaga interogare. Este logic, deoarece sistemul RDBMS nu știe ce alte rezultate ați dori să obțineți, decât dacă-i spuneți acest lucru – și tocmai acesta este scopul clauzei GROUP BY. Aceasta cere sistemului DBMS *să grupeze* rândurile selectate de interogare pe baza valorilor din una sau mai multe coloane și să aplice funcția (sau funcțiile) de agregare fiecărui grup, returnând un rând pentru fiecare grup din setul de rezultate. Este ca și cum ați cere subtotaluri pentru fiecare departament, în locul unui singur total pentru întreaga companie, dar, așa cum ați văzut, funcțiile de agregare pot face multe alte lucruri, nu numai însumarea unor valori. Sistemul DBMS va ordona rândurile selectate de interogare după coloanele din clauza GROUP BY, așa că grupurile vor fi returnate în ordine ascendentă, exceptând cazul în care adăugați o clauză ORDER BY care specifică un alt mod de ordonare. Iată un exemplu:

- Afișați fiecare cod de gen, împreună cu numărul de filme asociate fiecărui cod.

```
SELECT MOVIE_GENRE_CODE AS GENRE, COUNT(*) AS COUNT
  FROM MOVIE
 GROUP BY MOVIE_GENRE_CODE;

GENRE   COUNT
-----  ------
ActAd       7
Comdy       4
Drama       6
Forgn       1
Rmce        2
```

Ce se întâmplă dacă scoateți clauza GROUP BY din această interogare? Sistemul DBMS returnează un mesaj de eroare și, din nefericire, mesajul de eroare este deseori destul de criptic. Mulți programatori SQL începători au dificultăți în înțelegerea acestei probleme, dar fără clauza GROUP BY, interogarea este ilogică. Funcția COUNT(*) este o funcție de agregare și, în absența clauzei GROUP BY, returnează un singur rând de date. Pe de altă parte, MOVIE_GENRE_CODE este o coloană și, dacă nu este specificată în clauza GROUP BY, returnează o valoare pentru fiecare rând din tabel. Fără clauza GROUP BY care să relaționeze cele două elemente ale listei SELECT, sistemul DBMS nu știe ce să facă. Nici nu este de mirare că unii le numesc „funcții de agravare" în loc de „funcții de agregare". În realitate, acestea nu sunt chiar atât de greu de folosit. Rețineți o singură regulă: atunci când o interogare conține o funcție de agregare, *toate* coloanele din rezultatele interogării trebuie să fie formate folosind o funcție de agregare sau să fie specificate în clauza GROUP BY.

Operatori pentru interogări compuse

Uneori este util să rulați interogări multiple și să combinați rezultatele într-un singur set de rezultate.

UNION

Operatorul UNION adaugă rândurile din setul de înregistrări al unei interogări la cel al unei alte înregistrări și, în același timp, elimină rândurile duplicate, într-un mod similar cu cel al cuvântului cheie DISTINCT. Operația este permisă numai dacă interogările sunt compatibile din punctul de vedere al uniunii, ceea ce înseamnă că au același număr de coloane și că tipurile de date ale coloanelor corespondente sunt compatibile. Iată un exemplu:

- Afișați pe o singură coloană toate valorile nenule pentru taxa de închiriere și taxa de întârziere din tabelul MOVIE_RENTAL.

```
SELECT RENTAL_FEE AS FEE
  FROM MOVIE_RENTAL
 WHERE RENTAL_FEE IS NOT NULL
UNION
SELECT LATE_OR_LOSS_FEE AS FEE
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL;

      FEE
---------
        4
        6
     6.25
    29.98
    29.99
```

Dacă acest exemplu vi se pare greu de vizualizat, rulați independent cele două interogări, apoi combinați-le pentru a vedea rezultatul. Veți observa că al doilea și al treilea rând sunt taxele de închiriere, găsite de prima interogare, în timp ce primul, al patrulea și al cincilea rând sunt taxele de întârziere, găsite de a doua interogare.

UNION ALL

UNION ALL funcționează la fel ca și operatorul UNION, exceptând faptul că rândurile duplicate *nu sunt* eliminate. Dacă repetați exemplul anterior folosind operatorul UNION ALL, veți obține 24 de rânduri în loc de 5.

INTERSECT

Operatorul INTERSECT găsește valorile selectate dintr-o interogare, care apar și într-o altă interogare. În esență, găsește intersecția valorilor din cele două interogări. Totuși, doar un număr mic de sisteme DBMS (cele mai importante fiind Oracle și DB2) implementează acest operator. Nu-l veți găsi în Microsoft SQL Server sau MySQL. Iată un exemplu:

- Există în tabelul MOVIE filme pentru care prețul pentru DVD este egal cu prețul pentru VHS? Setul de rezultate nu conține nici un rând, așa că răspunsul este nu.

```
SELECT RENTAL_FEE AS FEE
  FROM MOVIE_RENTAL
 WHERE RENTAL_FEE IS NOT NULL
INTERSECT
SELECT LATE_OR_LOSS_FEE AS FEE
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL

no rows selected
```

EXCEPT

EXCEPT este operatorul standard ANSI/ISO care găsește diferențele dintre două seturi de rezultate, returnând, în esență, valorile din prima interogare care nu apar în cea de-a doua interogare. Foarte puține sisteme DBMS implementează acest operator. În unele implementări, precum Oracle, operatorul se numește MINUS, nu EXCEPT.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. O instrucțiune SELECT fără o clauză WHERE
   a. Selectează toate coloanele din tabel sau vizualizare
   b. Returnează un mesaj de eroare
   c. Selectează toate rândurile din tabel sau vizualizare
   d. Afișează numai definiția tabelului sau a vizualizării
   e. Scrie întotdeauna rezultatele într-un fișier jurnal

2. În SQL, ordinea rândurilor din rezultatele interogării
   a. Este specificată de clauza SORTED BY
   b. Poate fi ascendentă sau descendentă pentru orice coloană
   c. În mod prestabilit este descendentă, dacă nu se specifică o altă ordine
   d. Este imprevizibilă dacă nu este specificată în interogare
   e. Poate fi specificată numai pentru coloanele din setul de rezultate al interogării
3. Operatorul BETWEEN
   a. Specifică un domeniu de valori care include și capetele
   b. Poate fi rescris folosind operatorii <= și NOT <=
   c. Poate fi rescris folosind operatorii <= și >=
   d. Selectează rândurile adăugate în tabel într-un anumit interval de timp
   e. Nu este inclus în standardul ISO/ANSI
4. Operatorul LIKE standard
   a. Folosește semne de întrebare drept caractere de înlocuire poziționale
   b. Folosește liniuțe de subliniere drept caractere de înlocuire poziționale
   c. Folosește liniuțe de subliniere drept caractere de înlocuire nepoziționale
   d. Folosește simboluri procent drept caractere de înlocuire poziționale
   e. Folosește simboluri procent drept caractere de înlocuire nepoziționale
5. O instrucțiune SQL care conține o funcție de agregare
   a. Poate conține și coloane calculate
   b. Poate conține și coloane obișnuite
   c. Trebuie să includă o clauză ORDER BY
   d. Trebuie să includă o clauză GROUP BY
   e. Nu poate include, în același timp, o clauză GROUP BY și o clauză ORDER BY
6. Când operatorii AND și OR sunt combinați în aceeași clauză WHERE
   a. Sistemul DBMS returnează un mesaj de eroare
   b. Operatorul AND are prioritate mai mare decât operatorul OR

c. Operatorul AND are prioritate mai mică decât operatorul OR

d. Parantezele sunt obligatorii

e. Parantezele sunt opționale

7. Sintaxa corectă pentru eliminarea valorilor nule din rezultatele interogării este

a. = NULL

b. NOT = NULL

c. <> NULL

d. IS NULL

e. IS NOT NULL

8. Funcțiile SQL standard pentru șiruri de caractere includ

a. UPPER

b. MIDDLE

c. LOWER

d. SUBSTR

e. EXISTS

9. Funcțiile SQL matematice standard includ

a. LENGTH

b. ROUND

c. CAST

d. MIN

e. ABS

10. Operatorul UNION

a. Elimină rândurile duplicate din setul de rezultate

b. Include rândurile duplicate în setul de rezultate

c. Combină două interogări într-o singură interogare de tip join

d. Combină seturile de rezultate a două interogări într-un singur set de rezultate

e. Este numit JOIN în unele implementări SQL

Scrieți instrucțiunile SQL pentru următoarele probleme.

11. Găsiți toate filmele din tabelul MOVIE pentru care MPAA_RATING_CODE are altă valoare decât „R".

12. Afișați titlurile și prețurile tuturor filmelor pentru care RETAIL_PRICE_DVD este cel puțin 19.99, dar nu mai mare de 29.99, ordonate crescător după preț.

13. Afișați toate filmele pentru care genul (MOVIE_GENRE_CODE) este Comdy și categoria MPAA (MPAA_RATING_CODE) este PG-13, împreună cu filmele pentru care genul este Drama și categoria este R.

14. Câte închirieri (tabelul MOVIE_RENTAL) nu au nici o valoare în coloana LATE_OR_LOSS_FEE?

15. Câte persoane au un nume de familie (PERSON_FAMILY_NAME) care conține litera „a", majusculă sau minusculă?

16. Afișați toate titlurile de filme care conțin cuvântul „the", cu sau fără literă mare.

17. Folosiți funcția SUM pentru a afla totalul valorilor din coloana RENTAL_FEE din tabelul MOVIE_RENTAL.

18. Afișați primele cinci caractere din numele de familie (PERSON_FAMILY_NAME) din tabelul PERSON, dar eliminați toate valorile duplicate din setul de rezultate?

19. Din tabelul MOVIE, afișați toate genurile (MOVIE_GENRE_CODE), împreună cu media prețurilor pentru DVD (RETAIL_PRICE_DVD) pentru fiecare gen, rotunjită la două poziții zecimale.

20. Afișați toate filmele (MOVIE_ID) care au fost închiriate (tabelul MOVIE_RENTAL), cu suma totală strânsă din taxele de închiriere (RENTAL_FEE) sau taxele de întârziere (LATE_OR_LOSS_FEE) pentru filmul respectiv. Sugestie: adunați valorile RENTAL_FEE și LATE_OR_LOSS_FEE, apoi însumați (SUM) rezultatul respectiv pentru fiecare valoare MOVIE_ID. Unele valori din coloana LATE_OR_LOSS_FEE sunt nule, așa că, dacă nu aveți o funcție care să înlocuiască valorile nule cu o altă valoare (zero, în acest caz), veți obține valori nule în rezultate. În Oracle, funcția se numește NVL, în Microsoft SQL Server se numește ISNULL, iar în MySQL se numește IFNULL. (Se pare că nu există o funcție echivalentă în DB2).

CAPITOLUL 5

Combinarea datelor din mai multe tabele

În capitolul 4 am discutat despre instrucțiuni SQL care selectează date dintr-un singur tabel. Deseori, este util să combinați într-o singură interogare date din tabele multiple. De exemplu, în listingul celor trei coloane ale tabelului MOVIE din figura 5-1, observați valorile afișate pentru coloana MOVIE_GENRE_CODE. Deși unele rânduri conțin valori a căror semnificație o puteți ghici cu ușurință, precum „Drama" și „SciFi", altele, cum ar fi „Forgn", „ActAd", „ChFam" și „Indep", nu sunt la fel de evidente.

Atunci când am proiectat baza de date pentru magazinul de produse video, am folosit deliberat coduri în locul descrierilor complete pentru genurile filmelor în tabelul MOVIE. Nu numai că acesta este un mod bun de proiectare a bazelor de date, conform regulilor de normalizare, dar reduce și dimensiunea fiecărui rând din tabelul MOVIE, folosind un cod de cinci caractere în locul unei descrieri care ar putea avea 100 de caractere. Mai mult, așa cum vă amintiți din discuția despre procesul de normalizare, din capitolul 1, am evitat anomalia de actualizare – dacă

MOVIE_ID	MOVIE_GENRE_CODE	MOVIE_TITLE
1	Drama	Mystic River
2	ActAd	The Last Samurai
3	Comdy	Something's Gotta Give
4	ActAd	The Italian Job
5	ActAd	Kill Bill: Vol. 1
6	ActAd	Pirates of the Caribbean: The Curse of the Black Pearl
7	Drama	Big Fish
8	ActAd	Man on Fire
9	ActAd	Master and Commander: The Far Side of the World
10	Drama	Lost in Translation
11	Rmce	Two Weeks Notice
12	Comdy	50 First Dates
13	Comdy	Matchstick Men
14	Drama	Cold Mountain
15	Drama	Road to Perdition
16	Comdy	The School of Rock
17	Rmce	13 Going on 30
18	Drama	Monster
19	ActAd	The Day After Tomorrow
20	Forgn	Das Boot

20 rows selected.

Figura 5-1 Listingul tabelului MOVIE (trei coloane)

se schimbă descrierea unui gen, nu este nevoie să actualizez acea descriere pentru toate filmele asociate genului respectiv în tabelul MOVIE. În timpul normalizării, descrierea genurilor a fost mutată în propriul tabel, MOVIE_GENRE, iar coloana MOVIE_GENRE_CODE a devenit cheie externă în tabelul MOVIE, referind coloana cheie primară (cu același nume) din tabelul MOVIE_GENRE. În timp ce unii proiectanți preferă ca valorile cheilor primare să nu aibă nici o semnificație în lumea reală, deoarece este posibil ca valorile respective să se schimbe la un moment dat, eu am optat pentru folosirea unui cod mnemonic pentru codurile genurilor, deoarece astfel permit celor familiarizați cu datele respective să înțeleagă genul asociat filmelor fără să le caute în tabelul MOVIE_GENRE. Evident, nu puteți afișa pe pagina web a magazinului de produse video tabelul MOVIE așa cum este prezentat în figura 5-1 – trebuie să obțineți descrierea completă a genurilor din tabelul MOVIE_GENRE. Exact aceasta este ideea capitolului de față: combinarea datelor din mai multe tabele într-o singură interogare. Figura 5-2 prezintă un listing al tabelului MOVIE_GENRE.

MOVIE_GENRE_CODE	MOVIE_GENRE_DESCRIPTION
ActAd	Action and Adventure
Anime	Anime and Animation
ChFam	Children and Family
Class	Classics
Comdy	Comedy
Doc	Documentary
Drama	Drama
Forgn	Foreign
Hor	Horror
Indep	Independent
Music	Music and Musicals
Rmce	Romance
SciFi	Science Fiction and Fantasy
Specl	Special Interest
Sport	Sports
Thril	Thrillers

16 rows selected.

Figura 5-2 Listingul tabelului MOVIE_GENRE

Uniuni (join)

O *uniune (join)* este o operație într-o bază de date relaționale care combină coloane din două sau mai multe tabele în rezultatele unei singure interogări. O uniune apare de fiecare dată când clauza FROM a unei instrucțiuni SELECT specifică numele mai multor tabele. Iată un exemplu, dar, așa cum veți vedea, are o mare problemă:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
  FROM MOVIE, MOVIE_GENRE
 ORDER BY MOVIE_ID;

 MOVIE_ID   GENRE                 MOVIE_TITLE
---------   --------------------  ---------------
        1   Action and Adventure  Mystic River
        1   Anime and Animation   Mystic River
        1   Classics              Mystic River
        1   Documentary           Mystic River
        1   Foreign               Mystic River
        1   Independent           Mystic River
        1   Romance               Mystic River
        1   Special Interest      Mystic River
```

```
         1   Thrillers                Mystic River
         1   Sports                   Mystic River
         1   Science Fiction and      Mystic River
         1   Music and Musicals       Mystic River
         1   Horror                   Mystic River
         1   Drama                    Mystic River
         1   Comedy                   Mystic River
         1   Children and Family      Mystic River
         2   Action and Adventure     The Last Samurai
         2   Thrillers                The Last Samurai
         2   Sports                   The Last Samurai
         2   Special Interest         The Last Samurai
         2   Science Fiction and      The Last Samurai
         2   Romance                  The Last Samurai
         2   Music and Musicals       The Last Samurai
         2   Independent              The Last Samurai
         2   Horror                   The Last Samurai
         2   Anime and Animation      The Last Samurai
         2   Children and Family      The Last Samurai
         2   Documentary              The Last Samurai
         2   Foreign                  The Last Samurai
         2   Drama                    The Last Samurai
         2   Comedy                   The Last Samurai
         2   Classics                 The Last Samurai
       ...   ...                      ...
320 rows selected.
```

Setul de rezultate al interogării a fost trunchiat după primele două titluri de filme (32 de rânduri). Au fost selectat, în total, 320 de rânduri. Cum este posibil așa ceva, dacă în tabelul MOVIE există doar 20 de filme? Problema este că am cerut bazei de date să unească tabelele, dar nu i-am spus care este corespondența dintre rândurile celor două tabele. Rezultatul este cunoscut sub numele de *produs cartezian* (după numele filozofului și matematicianului francez René Descartes) și, în bazele de date relaționale, este un set de rezultate obținut prin uniunea dintre fiecare rând al unui tabel cu fiecare rând dintr-un alt tabel. În acest exemplu, am realizat o uniune între fiecare rând din tabelul MOVIE cu fiecare rând din tabelul MOVIE_GENRE, făcând să pară că fiecare film aparține tuturor genurilor, ceea ce este, desigur, incorect. Au rezultat 320 de rânduri deoarece există 16 genuri și 20 de filme (16 x 20 = 320). Puteți vedea acest model în setul de rezultate – primul film este asociat cu toate cele 16 genuri, apoi al doilea și așa mai departe.

Soluția este să specificați *cum* se va face unirea tabelelor. Mai precis, să indicați ce coloane ar trebui să se potrivească pentru a uni două rânduri din cele două tabele. În mod normal, cele două coloane vor fi cheia primară dintr-un tabel și cheia externă din celălalt tabel, dar pot exista și excepții. Probabil vă întrebați de ce limbajul SQL permite obținerea

unui produs cartezian, dar, chiar dacă vă vine greu să credeți, există situații în care acesta este necesar. Atenție – produsul cartezian a două tabele de 1000 de rânduri va returna un milion de rânduri! În secțiunea următoare veți învăța să faceți o uniune standard relațională, cunoscută și sub numele de *uniune de egalitate (equijoin)*.

Uniuni de egalitate (equijoin)

Avem o *uniune de egalitate (equijoin)*, numită și *uniune internă (inner join)*, atunci când legăm una sau mai multe coloane dintr-un tabel (de obicei, o cheie externă) cu coloane similare dintr-un alt tabel (de obicei, cheia primară), folosind condiția de *egalitate* (cu alte cuvinte, considerăm că acele coloane se potrivesc dacă valorile datelor sunt egale). Veți descoperi că aceasta este cea mai des folosită formă de uniune. Totuși, termenul *uniune de egalitate (equijoin)* este rareori folosit în afara mediilor academice, fiind preferate denumirile *uniune internă (inner join)* sau *uniune standard (standard join)*.

Există două modalități de specificare a coloanelor corespondente: folosind clauza WHERE sau folosind clauza JOIN. Clauza JOIN a fost adăugată relativ recent în standardul SQL, așa că programatorii mai vechi sunt obișnuiți cu metoda bazată pe clauza WHERE.

Realizarea uniunilor folosind clauza WHERE

Folosirea clauzei WHERE pentru unirea tabelelor seamănă cu folosirea acesteia pentru eliminarea rândurilor de care nu aveți nevoie din setul de rezultate. Totuși, există unele diferențe. În primul rând, în condiția WHERE comparați o coloană cu o altă coloană, nu o coloană cu o constantă sau o expresie. În al doilea rând, atunci când coloanele din cele două tabele au același nume (o soluție recomandată) trebuie să specificați numele complet al coloanelor (adică numele cu calificator), astfel încât motorul SQL să știe care dintre cele două coloane este referită. Deși pentru dumneavoastră poate părea evident ceea ce vreți să faceți, motorul SQL va insista să referiți fără ambiguități toate coloanele specificate într-o instrucțiune SQL. Aceasta nu înseamnă numai coloanele din clauza WHERE, ci și din orice alt loc al instrucțiunii, inclusiv în lista SELECT. Cea mai simplă formă de calificator este chiar numele tabelului, separat cu un caracter punct de numele coloanei. În continuare este prezentat un exemplu de uniune realizată prin clauza WHERE, cu numele coloanelor specificate complet, folosind numele tabelelor. Observați că interogarea selectează coloanele MOVIE_ID și MOVIE_TITLE din tabelul MOVIE și genul corespunzător (MOVIE_GENRE_DESCRIPTION) din tabelul MOVIE_GENRE.

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE, MOVIE_GENRE
 WHERE MOVIE.MOVIE_GENRE_CODE= MOVIE_GENRE.MOVIE_GENRE_CODE
 ORDER BY MOVIE_ID;
```

```
 MOVIE_ID   GENRE                MOVIE_TITLE
---------   -------------------  ---------------
        1   Drama                Mystic River
        2   Action and Adventure The Last Samurai
        3   Comedy               Something's Gotta Give
        4   Action and Adventure The Italian Job
        5   Action and Adventure Kill Bill: Vol. 1
        6   Action and Adventure Pirates of the Caribbean:
        7   Drama                Big Fish
        8   Action and Adventure Man on Fire
        9   Action and Adventure Master and Commander: The
       10   Drama                Lost in Translation
       11   Romance              Two Weeks Notice
       12   Comedy               50 First Dates
       13   Comedy               Matchstick Men
       14   Drama                Cold Mountain
       15   Drama                Road to Perdition
       16   Comedy               The School of Rock
       17   Romance              13 Going on 30
       18   Drama                Monster
       19   Action and Adventure The Day After Tomorrow
       20   Foreign              Das Boot

20 rows selected.
```

Folosirea numelor complete ale tabelelor pentru specificarea coloanelor poate fi obositoare și consumatoare de timp, mai ales deoarece numele tabelelor pot avea 30 sau mai multe caractere în sistemele DBMS moderne. Din această cauză, în SQL este permisă folosirea pseudonimelor (*aliases*) pentru numele tabelelor. Acestea funcționează la fel ca și pseudonimele coloanelor din clauza SELECT, exceptând faptul că nu este folosit cuvântul cheie „AS" (în cele mai multe implementări SQL) – doar lăsați un spațiu între numele tabelului și pseudonim în lista FROM. Deși unii folosesc mnemonice pentru pseudonimele tabelelor, este mult mai des întâlnită folosirea secvențelor de majuscule (adică „A", „B", „C" și așa mai departe). După ce asociați un pseudonim unui nume de tabel în clauza FROM, *trebuie* să folosiți pseudonimul în locul numelui de tabel în întreaga instrucțiune SQL. Folosirea pseudonimelor în clauza SELECT poate părea puțin ciudată la început, deoarece folosiți un pseudonim înainte de a-l defini (clauza SELECT precede clauza FROM) și s-ar putea să vi se pare mai simplu să completați clauza FROM înainte de a completa lista de coloane din clauza SELECT atunci când scrieți instrucțiunile SQL.

Exemplul următor prezintă instrucțiunea anterioară, după adăugarea pseudonimelor pentru numele tabelelor. Deși nu era necesar, pseudonimele au fost adăugate și în lista de coloane din clauzele SELECT și ORDER BY, pentru a vă arăta cum sunt folosite.

```
SELECT A.MOVIE_ID, B.MOVIE_GENRE_DESCRIPTION AS GENRE,
       A.MOVIE_TITLE
  FROM MOVIE A, MOVIE_GENRE B
 WHERE A.MOVIE_GENRE_CODE = B.MOVIE_GENRE_CODE
 ORDER BY A.MOVIE_ID;
```

Realizarea uniunilor folosind clauza JOIN

Așa cum am menționat deja, clauza JOIN este o nouă adăugire în SQL, fiind introdusă în standardul SQL-92, așa că este posibil ca implementarea pe care o folosiți să nu o accepte încă, deși este inclusă în majoritatea implementărilor moderne. Clauza JOIN este scrisă ca o referință de tabel în clauza FROM și, în esență, combină lista de tabele din clauza FROM și condiția de legătură scrisă anterior în clauza WHERE într-o singură clauză. Iată sintaxa generală a clauzei JOIN pentru o uniune internă, urmată de câteva exemple.

```
nume_tabel [INNER] JOIN nume_tabel
  { ON condiţie | USING (nume_coloană [,nume_coloană...]) }
```

Observați cele două opțiuni. Clauza ON permite specificarea unei condiții similare cu cea din clauza WHERE folosită în exemplul anterior. Clauza USING, pe de altă parte, specifică numele coloanelor folosite pentru legarea rândurilor. Totuși, clauza USING funcționează numai atunci când coloanele pe care se face legătura au nume identice în ambele tabele. Iată câteva exemple:

- JOIN cu condiție ON:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE JOIN MOVIE_GENRE ON
       MOVIE.MOVIE_GENRE_CODE = MOVIE_GENRE.MOVIE_GENRE_CODE
 ORDER BY MOVIE_ID;
```

- JOIN cu pseudonime în loc de nume de tabele:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE A JOIN MOVIE_GENRE B ON
       A.MOVIE_GENRE_CODE = B.MOVIE_GENRE_CODE
 ORDER BY MOVIE_ID;
```

- JOIN folosind cuvântul cheie USING (în locul condiției ON) – o scurtătură elegantă atunci când coloanele din cele două tabele au același nume. Totuși, această sintaxă este foarte nouă și, în momentul scrierii acestei cărți, era acceptată numai de Oracle și MySQL (nu și de Microsoft SQL Server și DB2).

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE JOIN MOVIE_GENRE USING (MOVIE_GENRE_CODE)
 ORDER BY MOVIE_ID;
```

- JOIN cu cheie externă pe mai multe coloane. Această interogare afișează lista cu copiile filmelor din tabelul MOVIE_COPY care nu au fost vândute (coloana DATE_SOLD conține o valoare nulă) și care au fost închiriate (uniune cu tabelul MOVIE_RENTAL) dar nu au fost încă returnate (coloana RETURNED_DATE conține o valoare nulă). Deoarece acest tabel nu a mai fost folosit în exemple, puteți revedea prezentarea generală a bazei de date a magazinului de produse video, pentru a vedea unde se încadrează tabelul în modelul general. De asemenea, rețineți că, în momentul scrierii cărții, numai Oracle și MySQL acceptau această sintaxă.

```
SELECT MOVIE_ID, COPY_NUMBER, DUE_DATE
  FROM MOVIE_COPY JOIN MOVIE_RENTAL
       USING (MOVIE_ID, COPY_NUMBER)
 WHERE DATE_SOLD IS NULL
   AND RETURNED_DATE IS NULL;

MOVIE_ID  COPY_NUMBER  DUE_DATE
--------  -----------  ----------
       2            2  02/27/2005
       3            2  03/04/2005
       5            1  02/27/2005
       5            2  02/19/2005
      10            1  03/04/2005
      17            1  03/04/2005

6 rows selected.
```

- Doar pentru a vă arăta cu cât sunt mai scurte instrucțiunile care folosesc clauza JOIN, iată interogarea precedentă scrisă folosind condiții de legătură în clauza WHERE. Remarcați că în acest caz trebuie să specificați numele complet al coloanelor MOVIE_ID și COPY_NUMBER, pentru a evita referirea ambiguă a lor. Instrucțiunea de mai jos afișează aceleași rezultate ca și instrucțiunea anterioară.

```
SELECT A.MOVIE_ID, A.COPY_NUMBER, DUE_DATE
  FROM MOVIE_COPY A, MOVIE_RENTAL B
 WHERE A.MOVIE_ID = B.MOVIE_ID
   AND A.COPY_NUMBER = B.COPY_NUMBER
   AND DATE_SOLD IS NULL
   AND RETURNED_DATE IS NULL;
```

Uniuni naturale

O *uniune naturală (natural join)* se bazează pe toate coloanele cu același nume din două tabele. În esență, toate uniunile de egalitate pe care le-ați văzut deja sunt uniuni naturale. Totuși, sintaxa unei uniuni naturale este mult mai simplă, deoarece nu este necesar să specificați o condiție sau o listă de coloane – se înțelege de la sine care sunt coloanele folosite. Rețineți că, în momentul scrierii cărții, numai Oracle și MySQL acceptau această sintaxă pentru clauza JOIN.

Unele descrieri ale uniunilor naturale, pe care s-ar putea să le găsiți în alte surse, precizează că lista de coloane selectate are legătură cu definiția unei uniuni naturale. De fapt, nu există nici o legătură – puteți selecta orice coloane doriți, dar, ca în orice instrucțiune SELECT, în setul de rezultate nu pot exista două coloane cu același nume, așa că trebuie să redenumiți coloanele duplicate folosind pseudonime.

Iată uniunea tabelelor MOVIE și MOVIE_GENRE, rescrisă sub forma unei uniuni naturale:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE NATURAL JOIN MOVIE_GENRE
 ORDER BY MOVIE_ID;
```

Uniunile pot implica mai mult de două tabele. Exemplul următor prezintă o uniune naturală care obține coloana MOVIE_ID din tabelul MOVIE, coloana MOVIE_GENRE_DESCRIPTION din tabelul MOVIE_GENRE și coloana MPAA_RATING_DESCRIPTION din tabelul MPAA_RATING, pentru primele cinci filme din tabelul MOVIE. Remarcați folosirea clauzei WHERE pentru a elimina din setul de rezultate rândurile de care nu avem nevoie. Am permis textului din coloana RATING_DESC să treacă automat pe linia următoare în rezultatele interogării, din cauza lungimii. De fapt, exemplul folosește două clauze JOIN. Prima clauză JOIN cere motorului SQL să lege tabelele MOVIE și MOVIE_GENRE, iar a doua clauză JOIN îi cere să lege rândurile deja unite (în esență, un set de rezultate intermediar) cu tabelul MPAA_RATING.

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE,
       MPAA_RATING_CODE AS RATING,
       MPAA_RATING_DESCRIPTION AS RATING_DESC
  FROM  MOVIE NATURAL JOIN MOVIE_GENRE
              NATURAL JOIN MPAA_RATING
  WHERE MOVIE_ID < 6
  ORDER BY MOVIE_ID;

MOVIE_ID  GENRE                RATING  RATING_DESC
--------  -------------------- ------  -------------------------
       1  Drama                R       Under 17 requires
                                       accompanying parent or
                                       adult guardian
```

```
         2  Action and Adventure  R       Under 17 requires
                                          accompanying parent or
                                          adult guardian
         3  Comedy                PG-13   Parents strongly cau
                                          tioned
         4  Action and Adventure  PG-13   Parents strongly cau
                                          tioned
         5  Action and Adventure  R       Under 17 requires
                                          accompanying parent or
                                          adult guardian

5 rows selected.
```

Dacă folosiți o implementare SQL care nu acceptă uniunile naturale, iată aceeași interogare scrisă folosind cuvintele cheie JOIN și ON:

```
SELECT A.MOVIE_ID, B.MOVIE_GENRE_DESCRIPTION AS GENRE,
       C.MPAA_RATING_CODE AS RATING,
       C.MPAA_RATING_DESCRIPTION AS RATING_DESC
  FROM MOVIE A JOIN MOVIE_GENRE B ON
       A.MOVIE_GENRE_CODE = B.MOVIE_GENRE_CODE
              JOIN MPAA_RATING C ON
       A.MPAA_RATING_CODE = C.MPAA_RATING_CODE
 WHERE MOVIE_ID < 6
 ORDER BY MOVIE_ID;
```

Uniuni externe (outer joins)

Toate uniunile pe care le-am văzut până acum sunt uniuni exclusive (un termen care-mi place mai mult decât *uniuni interne*), ceea ce înseamnă că singurele rânduri care apar în setul de rezultate sunt cele pentru care a fost găsită o legătură în toate tabelele unite. Totuși, există situații în care vreți să includeți în rezultatele interogării și rânduri pentru care nu există o legătură. De exemplu, ce se întâmplă dacă proprietarul magazinului de produse video vrea o listă cu toate genurile de filme, împreună cu toate titlurile asociate cu fiecare gen? Au existat clienți care au întâlnit probleme asemănătoare și care au cerut o soluție producătorilor de baze de date (și în final au obținut-o).

O *uniune externă (outer join)* – pentru care un nume mai potrivit ar fi *uniune inclusivă* – include în setul de rezultate și rândurile pentru care nu există legături din cel puțin unul dintre tabele. Atunci când există rânduri fără legături, datele selectate din tabelul în care nu a fost găsită o legătură primesc valoarea nulă. Există trei tipuri de bază:

- **Uniune externă către stânga (left outer join)** Returnează toate rândurile din tabelul din stânga (cel specificat primul în clauza JOIN), împreună cu toate rândurile din tabelul din dreapta pentru care poate fi găsită o legătură.

- **Uniune externă către dreapta (right outer join)** Returnează toate rândurile din tabelul din dreapta (cel specificat al doilea în clauza JOIN), împreună cu toate rândurile din tabelul din stânga pentru care poate fi găsită o legătură. În esență, o uniune externă către stânga poate fi rescrisă ca o uniune externă către dreapta inversând ordinea de specificare a tabelelor și înlocuind cuvântul cheie LEFT cu RIGHT.
- **Uniune externă completă (full outer join)** Returnează toate rândurile din ambele tabele. Acest tip de uniune este cel mai puțin probabil să fie acceptat de implementarea SQL pe care o aveți, deoarece sintaxa ei standard este mai nouă decât a celorlalte două. Este esențial să înțelegeți că această uniune nu este același lucru cu un produs cartezian, care leagă *fiecare* rând dintr-un tabel cu *fiecare* rând din celălalt tabel. O uniune externă completă (full outer join) leagă fiecare rând dintr-un tabel cu zero sau mai multe rânduri *corespondente* din celălalt tabel. În realitate, nu veți întâlni prea multe situații în care să folosiți o uniune externă completă, dar aceasta poate fi utilă dacă între două tabele aveți o relație opțională în ambele direcții.

Sintaxa generală pentru o uniune externă este

```
nume_tabel {RIGHT | LEFT | FULL} [OUTER] JOIN nume_tabel
 { ON condiție | USING (nume_coloană [,nume_coloană...]) }
```

Iată câteva exemple de uniuni externe:

- Afișați toate descrierile genurilor de filme, împreună cu filmele asociate fiecărui gen. Observați în setul de rezultate rândurile care nu au nici o valoare în coloana MOVIE_TITLE – acestea sunt genurile de filme care nu au filme asociate. Dacă implementarea DBMS nu acceptă uniuni realizate cu cuvântul cheie USING, modificați instrucțiunea astfel încât să utilizeze cuvântul cheie ON, ca în exemplul care urmează după acesta.

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE_GENRE LEFT OUTER JOIN MOVIE
       USING (MOVIE_GENRE_CODE);

GENRE                          MOVIE_TITLE
-----------------------------  ----------------------------
Action and Adventure           Kill Bill: Vol. 1
Action and Adventure           Man on Fire
Action and Adventure           Pirates of the Caribbean:
Action and Adventure           Master and Commander: The
Action and Adventure           The Day After Tomorrow
Action and Adventure           The Last Samurai
Action and Adventure           The Italian Job
```

```
Anime and Animation
Children and Family
Classics
Comedy                          50 First Dates
Comedy                          Matchstick Men
Comedy                          The School of Rock
Comedy                          Something's Gotta Give
Documentary
Drama                           Big Fish
Drama                           Road to Perdition
Drama                           Mystic River
Drama                           Monster
Drama                           Cold Mountain
Drama                           Lost in Translation
Foreign                         Das Boot
Horror
Independent
Music and Musicals
Romance                         13 Going on 30
Romance                         Two Weeks Notice
Science Fiction and Fantasy
Special Interest
Sports
Thrillers

31 rows selected.
```

- Interogarea anterioară, rescrisă ca uniune externă către dreapta, cu condiție ON.

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE RIGHT OUTER JOIN MOVIE_GENRE
       ON MOVIE.MOVIE_GENRE_CODE =
          MOVIE_GENRE.MOVIE_GENRE_CODE;
```

- Interogarea anterioară, rescrisă ca uniune externă completă. Deoarece fiecare film trebuie să aibă asociat un gen, această interogare va produce aceleași rezultate ca și uniunea externă către dreapta din interogarea anterioară. Totuși, dacă genurile filmelor ar fi opționale, uniunea externă completă v-ar fi permis să afișați atât genurile care nu au filme asociate, cât și filmele care nu au genuri asociate.

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE,
       MOVIE_TITLE
  FROM MOVIE FULL OUTER JOIN MOVIE_GENRE
       ON MOVIE.MOVIE_GENRE_CODE =
          MOVIE_GENRE.MOVIE_GENRE_CODE;
```

- Afișați descrierile genurilor care nu au filme asociate. Există și alte moduri de a face acest lucru, dar exemplul de față folosește faptul că rândurile pentru care nu există legături returnează valori nule pentru a selecta numai genurile care nu au filme asociate. Deoarece coloane MOVIE_TITLE va avea întotdeauna valori nule în setul de rezultate, nu avem nici un motiv să o selectăm (și să o afișăm).

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE
  FROM MOVIE RIGHT OUTER JOIN MOVIE_GENRE
       ON MOVIE.MOVIE_GENRE_CODE =
          MOVIE_GENRE.MOVIE_GENRE_CODE
 WHERE MOVIE_TITLE IS NULL;

GENRE
------------------------------
Anime and Animation
Children and Family
Classics
Documentary
Horror
Independent
Music and Musicals
Science Fiction and Fantasy
Special Interest
Sports
Thrillers

11 rows selected.
```

Ca răspuns la cererile clienților, mai mulți producători de baze de date relaționale au introdus uniunile externe înainte de standardizarea clauzei JOIN. Secțiunile următoare prezintă câteva implementări specifice ale sintaxei uniunii externe. Aceste soluții particulare sunt prezentate aici numai pentru că s-ar putea să le întâlniți în instrucțiuni SQL mai vechi. Ar trebui să folosiți clauza JOIN standard ori de câte ori este posibil, deoarece este portabilă și mai ușor de înțeles.

Sintaxa uniunii externe în Oracle

Oracle Corporation a decis să folosească un semn plus încadrat de paranteze pentru definirea uniunilor externe. Simbolul „(+)" este plasat în clauza WHERE în partea *opusă* tabelului din care vreți să fie returnate toate rândurile (indiferent dacă există legături pentru ele în celălalt tabel). Deoarece datele din celălalt tabel (cel din care sunt returnate numai rândurile pentru care există legături) sunt completate cu valori nule atunci când nu este găsit un rând corespondent, este mai ușor să rețineți că simbolul „(+)" este plasat în partea clauzei

WHERE unde vreți să adăugați valori nule. Iată exemplul anterior rescris folosind sintaxa Oracle:

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE
  FROM MOVIE A, MOVIE_GENRE B WHERE A.MOVIE_GENRE_CODE(+) =
       B.MOVIE_GENRE_CODE
   AND MOVIE_TITLE IS NULL;
```

Sintaxa uniunii externe în Microsoft SQL Server

Microsoft SQL Server folosește operatorul „*=" în condiția WHERE pentru o uniune externă către stânga și operatorul „=*" pentru o uniune externă către dreapta. În ambele cazuri, este obligatoriu ca asteriscul și semnul egal să nu fie separate de spații. Iată exemplul anterior rescris folosind sintaxa Microsoft SQL Server (ca uniune externă către dreapta):

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE
  FROM MOVIE A, MOVIE_GENRE B
 WHERE A.MOVIE_GENRE_CODE =*
       B.MOVIE_GENRE_CODE
   AND MOVIE_TITLE IS NULL;
```

Auto-uniuni (self joins)

O *auto-uniune (self join)* este o uniune a unui tabel cu el însuși. Acest tip de uniune poate părea ciudat la prima vedere, dar uneori există relații în care cheia primară și cheia externă se află în același tabel. Acestea se numesc *relații recursive* și există o asemenea relație și în baza de date a magazinului de produse video. (Puteți revedea figura 1-8 din capitolul 1.) Tabelul EMPLOYEE are o coloană numită SUPERVISOR_PERSON_ID, care este o cheie externă către coloana PERSON_ID din același tabel. Este folosită pentru a lega fiecare angajat de șeful direct, care, desigur, este un alt angajat, ceea ce înseamnă că și șeful respectiv are o coloană în tabelul EMPLOYEE. Interogarea următoare afișează trei coloane din tabelul EMPLOYEE, inclusiv PERSON_ID și SUPERVISOR_PERSON_ID:

```
SELECT PERSON_ID, EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,
       SUPERVISOR_PERSON_ID
  FROM EMPLOYEE;

PERSON_ID   HOURLY_RATE  SUPERVISOR_PERSON_ID
---------   -----------  --------------------
        1            15
        2          9.75                     1
       10          9.75                     1
```

Aceste date arată că angajații 2 și 10 sunt subordonați angajatului 1, iar angajatul 1 nu este subordonat nimănui – am putea paria că este proprietarul sau administratorul magazinului. Nu este nici o surpriză că plata pe oră (HOURLY_RATE) a șefului este mai mare decât cea a subordonaților.

Acum să presupunem că magazinul trebuie să producă un raport cu diferențele de salariu dintre șefi și subordonați. Exemplul nostru conține puține date, așa că aceste calcule ar putea fi făcute manual, dar să presupunem că vreți să automatizați raportul și să-l distribuiți întregului lanț de magazine care folosește modelul acestei baze de date. Puteți lega înregistrarea fiecărui angajat de cea a șefului direct, astfel încât să obțineți plata pe oră a șefului. Iată interogarea care face acest lucru:

```
SELECT A.PERSON_ID, A.EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,
       B.EMPLOYEE_HOURLY_RATE AS SUPV_HOURLY_RATE
  FROM EMPLOYEE A JOIN EMPLOYEE B
       ON A.SUPERVISOR_PERSON_ID = B.PERSON_ID;

PERSON_ID   HOURLY_RATE    SUPV_HOURLY_RATE
---------   -----------    ----------------
         2          9.75                  15
        10          9.75                  15
```

Observați următoarele:

- Clauza FROM include de două ori tabelul EMPLOYEE. Puteți folosi și vechiul stil, cu clauză WHERE, pentru realizarea unei auto-uniuni. Iată cum ar arăta interogarea în acest caz:

```
SELECT A.PERSON_ID, A.EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,
       B.EMPLOYEE_HOURLY_RATE AS SUPV_HOURLY_RATE
  FROM EMPLOYEE A, EMPLOYEE B
 WHERE A.SUPERVISOR_PERSON_ID = B.PERSON_ID
```

- Numele tabelelor *trebuie* să includă pseudonime, deoarece la unirea unui tabel cu el însuși, *toate* numele de coloane sunt ambigue. Ca urmare, trebuie să prefixați numele fiecărei coloane cu pseudonimul tabelului.
- Administratorul magazinului (rândul cu PERSON_ID = 1) nu apare în rezultatele interogării. De ce? Dacă ați răspuns că aceasta este o uniune internă, ați ghicit. Angajatul 1 nu are nici un șef (coloana SUPERVISOR_PERSON_ID are o valoare nulă pentru PERSON_ID = 1), așa că uniunea internă nu găsește un rând corespondent și, ca rezultat, toate rândurile pentru care nu se găsește un corespondent sunt eliminate din setul de rezultate.

Iată și interogarea finală, care include calcularea diferenței de salariu și realizează o uniune cu tabelul PERSON pentru a obține numele angajaților:

```
SELECT A.PERSON_ID, C.PERSON_GIVEN_NAME AS FIRST_NAME,
       C.PERSON_FAMILY_NAME AS LAST_NAME,
       B.EMPLOYEE_HOURLY_RATE - A.EMPLOYEE_HOURLY_RATE
                              AS RATE_DIFF
  FROM EMPLOYEE A JOIN EMPLOYEE B
       ON A.SUPERVISOR_PERSON_ID = B.PERSON_ID
     JOIN PERSON C
       ON A.PERSON_ID = C.PERSON_ID;

PERSON_ID  FIRST_NAME  LAST_NAME   RATE_DIFF
---------  ----------  ----------  ---------
        2  Tin         Chung            5.25
       10  Gerald      Bernstein        5.25
```

Alte tipuri de uniuni

Cele mai multe uniuni sunt uniuni de egalitate. Totuși, nu este obligatoriu să folosiți întotdeauna o condiție de egalitate pentru verificarea corespondenței între rândurile unite. Atenție, totuși, deoarece uniunile care nu sunt uniuni de egalitate pot implica probleme de performanță, deoarece este mai puțin probabil ca sistemul DBMS să folosească indexuri atunci când operatorul condițional este altceva decât „=".

O utilizare frecventă a operatorului „diferit" (<>) în uniuni este căutarea rândurilor duplicate dintr-un tabel. Atunci când uniți un tabel cu el însuși pentru căutarea duplicatelor, trebuie să adăugați o condiție de legătură prin care să vă asigurați că rândurile nu sunt legate cu ele însele. Iată o interogare care caută titlurile duplicate din tabelul MOVIE. Observați comparația „diferit" (<>) pentru cheia primară (MOVIE_ID), care împiedică legarea unui rând de el însuși:

```
SELECT A.MOVIE_ID, B.MOVIE_ID
  FROM MOVIE A JOIN MOVIE B
       ON A.MOVIE_ID <> B.MOVIE_ID
      AND A.MOVIE_TITLE = B.MOVIE_TITLE;
```

Uniuni încrucișate (cross joins)

O *uniune încrucișată (cross join)* nu este altceva decât sintaxa standard pentru un produs cartezian. Așa cum am menționat anterior, există situații în care doriți să obțineți un produs cartezian și probabil astfel de situații au reprezentat motivul pentru care cei din comisiile de standardizare au adăugat această sintaxă în limbajul SQL standard. Interogarea care apărea la începutul capitolului (care unea tabelele MOVIE și MOVIE_GENRE și obținea un produs cartezian) poate fi rescrisă ca mai jos, folosind o uniune încrucișată:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
  FROM MOVIE CROSS JOIN MOVIE_GENRE
 ORDER BY MOVIE_ID;
```

Subinterogări

O caracteristică foarte puternică a limbajului SQL sunt *subinterogările* (numite și *subselecții*), care, așa cum sugerează și numele, se referă la o instrucțiune SELECT care conține o instrucțiune SELECT subordonată. De obicei, subinterogările sunt folosite în clauza WHERE, ca modalitate de limitare a rândurilor returnate în setul de rezultate al interogării externe. Aceasta poate fi o modalitate foarte flexibilă de selectare a datelor. Totuși, subinterogările pot fi folosite și în alte scopuri, așa cum veți vedea în secțiunea „Vizualizări în linie".

O regulă esențială de sintaxă este ca subinterogarea să fie încadrată de paranteze. O altă idee importantă pe care trebuie să o înțelegeți este că orice operație efectuată cu o subinterogare poate fi făcută și printr-o uniune. De fapt, unele implementări SQL rescriu automat subinterogările ca uniuni în interiorul motorului SQL înainte de a le rula, dar, exceptând cazul în care studiați implementarea internă a motorului SQL, ca utilizator al bazei de date nu veți ști niciodată că s-a întâmplat așa ceva.

Subinterogări necorelate

O *subinterogare necorelată (noncorrelated subselect)* este o subinterogare în care interogarea internă nu face nici o referire la interogarea externă care o conține. Aceasta înseamnă că poate fi rulată mai întâi interogarea internă, apoi setul de rezultate obținut poate fi folosit în interogarea externă. Iată câteva exemple:

- Afișați toate limbile în care nu există nici un film în inventarul magazinului de produse video.

```
SELECT LANGUAGE_CODE, LANGUAGE_NAME
  FROM LANGUAGE
 WHERE LANGUAGE_CODE NOT IN
   (SELECT DISTINCT LANGUAGE_CODE
      FROM MOVIE_LANGUAGE)
 ORDER BY LANGUAGE_CODE

LANGUAGE_CODE  LANGUAGE_NAME
-------------  -------------
ja             Japanese
ko             Korean
nl             Dutch
ru             Russian
zh             Chinese
```

Interogarea internă returnează o listă cu codurile de limbă asociate filmelor în tabelul MOVIE_LANGUAGE. Cuvântul cheie DISTINCT elimină rândurile

duplicate din setul de rezultate și, astfel, face ca interogarea externă să fie mult mai eficientă – cu cât apar mai puține valori în lista clauzei IN, cu atât este mai eficientă interogarea. În tabelul MOVIE_LANGUAGE există 40 de rânduri și, care urmare, lista IN ar trebui să conțină 40 de valori, dar cuvântul cheie DISTINCT reduce lista IN la numai 4 valori. Iată interogarea internă rulată independent, pentru a vă ajuta să înțelegeți cum funcționează:

```
SELECT DISTINCT LANGUAGE_CODE
  FROM MOVIE_LANGUAGE;

LANGUAGE_CODE
-------------
de                (German)
en                (English)
es                (Spanish)
fr                (French)
```

- Proprietarul magazinului vrea să vadă efectul recentei creșteri de prețuri și are nevoie de o listă a tranzacțiilor (TRANSACTION_ID) în care clientul a plătit mai mult decât taxa medie (RENTAL_FEE) pentru un film. Iată cum arată interogarea:

```
SELECT DISTINCT TRANSACTION_ID
  FROM MOVIE_RENTAL
 WHERE RENTAL_FEE >
   (SELECT AVG(RENTAL_FEE)
      FROM MOVIE_RENTAL)

TRANSACTION_ID
--------------
             9
            10
```

Interogarea internă calculează media taxelor de închiriere, iar interogarea externă găsește apoi toate rândurile din tabelul MOVIE_RENTAL pentru care valoarea RENTAL_FEE depășește media. Deși IN și NOT IN sunt operatorii folosiți cel mai des pentru conectarea subinterogărilor cu interogările externe, în acest caz subinterogarea returnează un singur rând; ca urmare puteți folosi operatorul mai mare decât (>) pentru comparare. Cuvântul cheie DISTINCT elimină tranzacțiile duplicate.

- Rezultatele interogării anterioare ar fi mai utile pentru proprietarul magazinului dacă ar include și data tranzacției. O modalitate de a realiza acest lucru este să adăugăm o uniune cu tabelul CUSTOMER_TRANSACTION în interogarea externă. Totuși, deoarece tocmai ați învățat despre subinterogări, să facem acest lucru folosind încă o subinterogare. Iată cum arată interogarea:

```
SELECT TRANSACTION_ID, TRANSACTION_DATE AS TRANS_DATE
  FROM CUSTOMER_TRANSACTION
 WHERE TRANSACTION_ID IN
   (SELECT DISTINCT TRANSACTION_ID
      FROM MOVIE_RENTAL
     WHERE RENTAL_FEE >
      (SELECT AVG(RENTAL_FEE)
         FROM MOVIE_RENTAL))

TRANSACTION_ID  TRANS_DATE
--------------  ----------
             9  03/01/2005
            10  03/01/2005
```

Subinterogări corelate

O *subinterogare corelată (correlated subselect)* este o subinterogare în care interogarea internă referă valorile furnizate de interogarea externă. Aceste subinterogări sunt mult mai puțin eficiente decât subinterogările necorelate, deoarece interogarea internă trebuie să fie apelată pentru fiecare rând găsit de interogarea externă. Vă amintiți că în cazul unei subinterogări necorelate, interogarea internă este rulată o singură dată. Iată un exemplu:

- Proprietarul magazinului vrea să transmită prin poștă un cupon valoric tuturor clienților care au plătit mai mult de 15$ pentru o singură tranzacție de închiriere. Iată cum arată interogarea:

```
SELECT DISTINCT CUSTOMER_ACCOUNT_ID
  FROM CUSTOMER_TRANSACTION A
 WHERE 15 <
   (SELECT SUM(RENTAL_FEE)
      FROM MOVIE_RENTAL B
     WHERE A.TRANSACTION_ID = B.TRANSACTION_ID)

CUSTOMER_ACCOUNT_ID
-------------------
                  2
                  7
                  9
```

Observați pseudonimele asociate numelor de tabele din interogările internă și externă, precum și folosirea acestora în clauza WHERE a interogării interne. Acesta este elementul de identificare al unei subinterogări corelate. Interogarea externă selectează o listă de valori CUSTOMER_ACCOUNT_ID distincte din tabelul CUSTOMER_TRANSACTION. Fiecare valoare găsită este transmisă interogării interne, care este rulată pentru a calcula

suma taxelor de închiriere din tranzacția respectivă. Dacă suma taxelor de închiriere este mai mare sau egală cu 15 (de fapt, condiția din interogare este „dacă 15 este mai mic decât suma taxelor de închiriere"), atunci clauza WHERE din interogarea externă ia valoarea logică „adevărat", iar rândul CUSTOMER_ID este adăugat în setul de rezultate.

Vizualizări în linie

Foarte puține implementări, printre care Oracle, permit folosirea unei subinterogări în clauza FROM a unei interogări, într-o construcție numită *vizualizare în linie (inline view)*. În esență, această construcție permite care setul de rezultate al unei interogări să fie tratat ca și cum ar fi un tabel sau o vizualizare predefinită. Iată un exemplu:

- Această interogare află numărul maxim de închirieri ale unui singur film:

```
SELECT MAX(RENTAL_COUNT) AS MAX_RENTAL_COUNT
  FROM
    (SELECT MOVIE_ID, COUNT(*) AS RENTAL_COUNT
       FROM MOVIE_RENTAL
      GROUP BY MOVIE_ID)

MAX_RENTAL_COUNT
----------------
               5
```

Interogarea internă calculează numărul de închirieri pentru fiecare valoare MOVIE_ID. Interogarea externă selectează valoarea maximă RENTAL_COUNT din interogarea internă, tratată ca și cum ar fi o vizualizare predefinită. Nu există nici o limită în privința modurilor de utilizare a vizualizărilor în linie – puteți chiar să le uniți cu alte tabele sau vizualizări.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. O subinterogare
   a. Poate fi folosită pentru a selecta valorile care vor fi aplicate condițiilor din clauza WHERE
   b. Poate fi corelată sau necorelată
   c. Reprezintă o cale puternică de calculare a coloanelor
   d. Trebuie să nu fie încadrate în paranteze
   e. Permite selectarea flexibilă a rândurilor

2. O uniune (*join*) fără o clauză WHERE sau o clauză JOIN
   a. Are ca rezultat un mesaj de eroare
   b. Nu returnează nici un rând din setul de rezultate
   c. Reprezintă o uniune externă (outer join)
   d. Reprezintă o uniune internă (inner join)
   e. Are ca rezultat un produs cartezian
3. O uniune externă (*outer join*)
   a. Poate fi scrisă în Oracle SQL folosind un simbol (+) în clauza FROM
   b. Poate fi scrisă în Microsoft SQL Server folosind operatorul *= sau =* în clauza WHERE
   c. Returnează toate rândurile doar din unul dintre tabele
   d. Returnează toate rândurile din unul sau din ambele tabele
   e. Poate fi către stânga, către dreapta sau completă
4. O auto-uniune (*self join*)
   a. Nu poate avea niciodată ca rezultat un produs cartezian
   b. Poate fi o uniune internă sau o uniune externă
   c. Rezolvă relațiile recursive
   d. Poate folosi o subinterogare pentru a limita și mai mult rândurile selectate
   e. Implică două tabele diferite
5. O uniune (*join*)
   a. Combină coloanele din două sau mai multe tabele în rezultatele unei singure interogări
   b. Combină rânduri din interogări multiple într-un singur set de rezultate
   c. Este realizată ori de câte ori în clauza FROM sunt specificate mai multe tabele
   d. Necesită folosirea unei clauze JOIN
   e. Necesită o listă de tabele separate prin virgule în clauza FROM

6. O uniune de egalitate (*equijoin*)
   a. Este cunoscută și sub numele de uniune externă (*outer join*)
   b. Este cunoscută și sub numele de uniune internă (*inner join*)
   c. Este cunoscută și sub numele de auto-uniune (*self join*)
   d. Realizează întotdeauna legarea rândurilor folosind o condiție de egalitate (=)
   e. Realizează întotdeauna legarea rândurilor folosind o condiție de inegalitate (<>)
7. Calificatorii numelor de coloane
   a. Rezolvă referințele ambigue la coloane
   b. Pot fi nume de tabele
   c. Pot fi numere care indică poziția relativă a tabelelor în lista FROM
   d. Pot fi pseudonime pentru numele de coloane, definite în clauza FROM
   e. Pot fi pseudonime pentru numele de tabele, definite în clauza FROM
8. O uniune încrucișată este același lucru cu
   a. O uniune naturală
   b. Un produs cartezian
   c. O uniune externă
   d. O uniune internă
   e. O auto-uniune
9. O clauză JOIN folosind cuvântul cheie USING
   a. Nu poate fi folosită atunci când coloanele prin care se face legarea tabelelor au nume diferite
   b. Nu poate fi folosită atunci când coloanele prin care se face legarea tabelelor au aceleași nume
   c. Definește o uniune internă
   d. Definește o uniune externă
   e. Definește o auto-uniune

10. O subinterogare corelată
    a. Rulează mai eficient decât o subinterogare necorelată
    b. Rulează mai puțin eficient decât o subinterogare necorelată
    c. Are o interogare internă care referă coloane din interogarea externă
    d. Are o interogare externă care referă coloane din interogarea internă
    e. Are o interogare internă care nu face nici o referire la coloanele din interogarea externă

Scrieți instrucțiunile SQL pentru următoarele probleme

11. Afișați numele și identificatorul de client ale tuturor persoanelor decedate (valoarea DEATH_DATE nu este nulă) care încă au conturi de clienți.

12. Afișați numele angajaților care au conturi de clienți. Folosiți uniuni pentru a stabili ce clienți sunt angajați și ce angajați au conturi de clienți.

13. Rescrieți interogarea anterioară pentru a folosi subinterogări în loc de uniuni.

14. Afișați numele de familie al fiecărui angajat, împreună cu numele de familie al șefului lor.

15. Afișați numărul de închirieri pentru fiecare format (DVD și VHS).

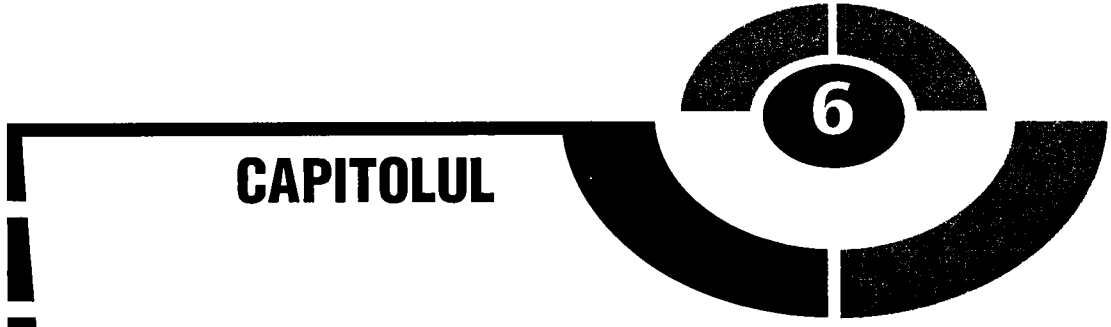
16. Afișați numărul de identificare (MOVIE_ID) și data la care trebuie returnate (DUE_DATE) pentru toate filmele închiriate și încă nereturnate (coloana RETURNED_DATE din tabelul MOVIE_RENTAL conține valori nule), dar care nu au fost cumpărate (coloana DATE_SOLD din tabelul MOVIE_COPY conține valori nule).

17. Afișați numărul de identificare și titlul filmelor care nu au fost închiriate niciodată.

18. Folosind subinterogări pentru a filtra rândurile, afișați titlul filmelor pentru care în inventar există o copie VHS (MEDIA_FORMAT = 'V', iar DATE_SOLD are valoarea NULL) dar pentru care nu există copii DVD (MEDIA_FORMAT = 'D').

19. Rescrieți interogarea anterioară folosind o uniune în locul unei subinterogări pentru a găsi filmele în format VHS.

20. Afișați toate filmele pentru care în inventar există mai multe copii în același format.



Scrierea interogărilor avansate

Înainte de a trece la limbajul de manipulare a datelor (DML), în capitolul 7, capitolul de față are rolul de a vă completa cunoștințele despre interogările SQL, prezentând câteva subiecte avansate pe care doar le-am menționat în capitolele 4 și 5. Vom discuta următoarele subiecte:

- Funcții SQL avansate, inclusiv funcții pentru caractere, matematice și pentru dată sau oră
- O descriere a modului în care puteți beneficia de avantajele vizualizărilor
- Informații despre expresia SQL CASE și despre utilizarea acesteia pentru construirea instrucțiunilor care includ porțiuni executate numai în anumite condiții predefinite

Funcții SQL avansate

Despre funcțiile SQL am discutat în capitolul 4. Secțiunile următoare descriu funcții care nu au fost prezentate în capitolul 4, dar pe care s-ar putea să le considerați utile. Pe lângă funcțiile pentru caractere și matematice, sunt incluse câteva funcții pentru date și ore. Rețineți că toate funcțiile SQL au o trăsătură comună, în sensul că returnează o singură valoare, așa că sunt utile în diferite locuri din instrucțiunile SQL, inclusiv în lista de coloane a comenzii SELECT și în clauza WHERE. Vă reamintim că există un număr mare de funcții specifice fiecărei implementări, furnizate de diferiți producători DBMS, așa că nu uitați să consultați documentația producătorilor pentru a găsi mai multe funcții utile.

Funcții pentru caractere

Funcțiile pentru caractere operează asupra datelor de tip text. Această secțiune prezintă câteva funcții frecvent folosite, pe lângă cele discutate în capitolul 4. Cu excepțiile menționate, funcțiile sunt acceptate în majoritatea implementărilor SQL răspândite, inclusiv Microsoft SQL Server, Oracle, DB2 și MySQL.

REPLACE

Funcția REPLACE caută un șir de caractere și înlocuiește caracterele găsite cu caracterele din șirul de înlocuire. Iată sintaxa generală a funcției:

```
REPLACE (şir_de_caractere, şir_căutat, şir_de_înlocuire)
```

- *șir_de_caractere* reprezintă șirul de caractere în care se face căutare și, în cele mai multe cazuri, este numele unei coloane dintr-un tabel, dar poate fi orice expresie care are ca rezultat un șir de caractere.
- *șir_căutat* este un șir format dintr-un caracter sau mai multe, care trebuie căutate în *șir_de_caractere*.
- *șir_de_înlocuire* este șirul de caractere cu care se înlocuiește orice apariție a *șirului_căutat* în *șir_de_caractere*.

Iată un exemplu care înlocuiește toate liniuțele de despărțire din numărul de telefon al unei persoane cu puncte (sunt prezentate numai primele două rânduri din setul de rezultate):

```
SELECT PERSON_PHONE,
       REPLACE(PERSON_PHONE,'-','.') AS DISPLAY_PHONE
  FROM PERSON;
```

```
PERSON_PHONE  DISPLAY_PHONE
------------  -------------
230-229-8976  230.229.8976
401-617-7297  401.617.7297
```

LTRIM

Funcția LTRIM elimină spațiile de la începutul unui șir de caractere. Rețineți că sunt eliminate *numai* spațiile de la începutul șirului – cele din mijlocul și de la sfârșitul șirului sunt lăsate ca atare. În baza de date a magazinului de produse video nu există șiruri de caractere cu spații la început și/sau la sfârșit, așa că vom folosi un exemplu general:

```
LTRIM (' String with spaces ')
Returns this string: 'String with spaces '
```

RTRIM

Funcția RTRIM este similară cu LTRIM, dar elimină spațiile de la sfârșitul unui șir de caractere. Dacă trebuie să eliminați atât spațiile de la începutul șirului, cât și pe cele de la sfârșit, puteți imbrica funcțiile LTRIM și RTRIM, ca în următorul exemplu:

```
RTRIM(LTRIM (' String with spaces '))
Returns this string: 'String with spaces'
```

NOTĂ: Oracle vă pune la dispoziție o funcție mai convenabilă, numită TRIM, care elimină atât spațiile de la începutul șirului, cât și pe cele de la sfârșit. În cazul altor implementări, puteți să imbricați funcțiile LTRIM și RTRIM și să obțineți același rezultat.

Funcții pentru valori nule (NVL, ISNULL, IFNULL)

Oracle, Microsoft SQL Server și MySQL pun la dispoziție o funcție care înlocuiește valorile nule cu o valoare specificată. Din nefericire, fiecare implementare folosește propriul nume pentru această funcție: NVL în Oracle, ISNULL în SQL Server și IFNULL în MySQL. Se pare că în DB2 nu există o funcție echivalentă. Exemplele următoare selectează coloana LATE_OR_LOSS_FEE din tabelul MOVIE_RENTAL, înlocuind valorile nule cu zero. A fost selectată tranzacția 9 deoarece conține două filme, dintre care unul are o valoare nulă în coloana LATE_OR_LOSS_FEE (un exemplu bun, care ilustrează faptul că valorile nule sunt transformate, în timp ce valorile nenule sunt lăsate ca atare.)

Oracle:

```
SELECT NVL(LATE_OR_LOSS_FEE, 0) AS LATE_OR_LOSS_FEE
  FROM MOVIE_RENTAL
 WHERE TRANSACTION_ID=9;

LATE_OR_LOSS_FEE
----------------
               0
           29.98

2 rows selected.
```

Microsoft SQL Server:

```
SELECT ISNULL(LATE_OR_LOSS_FEE, 0) AS LATE_OR_LOSS_FEE
  FROM MOVIE_RENTAL
 WHERE TRANSACTION_ID=9;

LATE_OR_LOSS_FEE
----------------
           29.98
             .00

(2 rows affected)
```

MySQL:

```
SELECT IFNULL(LATE_OR_LOSS_FEE, 0) AS LATE_OR_LOSS_FEE
  FROM MOVIE_RENTAL
 WHERE TRANSACTION_ID=9;

+------------------+
| LATE_OR_LOSS_FEE |
+------------------+
|            29.98 |
|             0.00 |
+------------------+

2 rows in set (0.16 sec)
```

Observați cât de diferită este formatarea rezultatelor pentru clientul MySQL în linia de comandă. Deși clienții SQL ai diferiților producători formatează, de obicei, în mod diferit rezultatele, vestea bună este că datele sunt aceleași.

ASCII

Funcția ASCII returnează valoarea din setul de caractere ASCII (un număr între 0 și 255) pentru un șir format dintr-un singur caracter. De exemplu, codul ASCII pentru spațiu este 32, așa că ASCII(' ') returnează valoarea 32.

CHAR (CHR)

Funcția CHAR (numită CHR în Oracle și DB2) returnează caracterul corespunzător unei valori ASCII (un număr între 0 și 255). De exemplu, funcția ASCII(44) returnează o virgulă, deoarece codul ASCII pentru virgulă este 44. Această funcție este utilă în special pentru concatenarea caracterelor care nu pot fi afișate sau sunt greu de manipulat în SQL. Câteva dintre caracterele ASCII folosite frecvent cu această funcție sunt prezentate în tabelul următor. Puteți folosi funcția ASCII sau un tabel cu setul de caractere ASCII (ușor de găsit în Internet) dacă vreți să aflați și alte valori.

Valoare ASCII	Caracter
9	Tab
10	Linie nouă
13	Retur de car (CR)
39	Apostrof

Urmează câteva exemple. Nu uitați că operatorul de concatenare nu este același pentru toate implementările DBMS ('+' pentru Microsoft SQL Server și 'll' pentru majoritatea celorlalte sisteme).

- Găsiți toate titlurile de filme care conțin un caracter Tab:

  Microsoft SQL Server:

```
SELECT MOVIE_ID FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%'+CHAR(9)+'%';

MOVIE_ID
(0 rows affected)
```

  Oracle și DB2:

```
SELECT MOVIE_ID FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%' || CHR(9) || '%';

no rows selected
```

NOTĂ: Puteți modifica interogarea anterioară astfel încât să găsească apostrofurile din titluri, în locul caracterelor Tab, înlocuind valoarea 9 cu 39. Ar trebui să găsiți filmul pentru care MOVIE_ID are valoarea 3 (Something's Gotta Give).

Funcții matematice

Așa cum probabil bănuiți din nume, funcțiile matematice returnează rezultatul unei operații matematice și, de obicei, acceptă ca parametru de intrare o expresie numerică, care poate fi o valoare literală, o valoare numerică dintr-o coloană a unui tabel sau orice expresie (inclusiv rezultatul unei alte funcții) care produce o valoarea numerică.

SIGN

Funcția SIGN primește ca argument o expresie numerică și returnează una dintre următoarele valori, în funcție de semnul numărului de intrare:

Valoare returnată	Semnificație
-1	Numărul de intrare este negativ
0	Numărul de intrare este zero
1	Numărul de intrare este pozitiv
null	Valoarea de intrare este nulă

Iată un exemplu:

```
SELECT LATE_OR_LOSS_FEE,
       SIGN(LATE_OR_LOSS_FEE) AS FEE_SIGN
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL;

LATE_OR_LOSS_FEE  FEE_SIGN
----------------  --------
           29.99         1
               4         1
               4         1
           29.98         1
```

SQRT

Funcția SQRT primește ca argument o expresie numerică și returnează rădăcina pătrată a acesteia. Sintaxa generală a funcției este

```
SQRT (expresie_numerică)
```

Rezultatul este puțin cam fără sens, dar vom extrage rădăcina pătrată din valorile LATE_OR_LOSS_FEE nenule pe care tocmai le-am văzut:

```
SELECT LATE_OR_LOSS_FEE,
       SQRT(LATE_OR_LOSS_FEE) AS FEE_SQRT
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL;

LATE_OR_LOSS_FEE     FEE_SQRT
----------------   ----------
           29.99   5.47631263
               4            2
               4            2
           29.98   5.47539953
```

CEILING (CEIL)

Funcția CEILING returnează cel mai mic întreg mai mare sau egal cu valoarea expresiei numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjește numărul prin adăugire până la următorul număr întreg. Există câteva probleme interesante de compatibilitate legate de denumire între implementările SQL: Microsoft SQL Server folosește numele CEILING, Oracle folosește numele CEIL, în timp ce DB2 și MySQL permite folosirea ambelor nume (CEIL și CEILING).

Ca exemplu, să aplicăm funcția CEILING asupra valorilor din coloana LATE_OR_LOSS_FEE (dacă folosiți Oracle, înlocuiți CEILING cu CEIL):

```
SELECT LATE_OR_LOSS_FEE,
       CEILING(LATE_OR_LOSS_FEE) AS FEE_CEILING
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL;

LATE_OR_LOSS_FEE  FEE_CEILING
----------------  -----------
            4.00            4
            4.00            4
           29.99           30
           29.98           30
```

FLOOR

Funcția FLOOR este inversa logică a funcției CEILING – returnează cel mai mare întreg mai mic sau egal cu valoarea expresiei numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjește numărul prin scădere până la următorul număr întreg.

Iată un exemplu, în care aplicăm funcția FLOOR asupra valorilor din coloana LATE_OR_LOSS_FEE:

```
SELECT LATE_OR_LOSS_FEE,
       FLOOR(LATE_OR_LOSS_FEE) AS FEE_FLOOR
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NOT NULL;

LATE_OR_LOSS_FEE   FEE_FLOOR
----------------   ---------
           4.00           4
           4.00           4
          29.99          29
          29.98          29
```

Funcții pentru dată și oră

Există foarte puțină consecvență în privința funcțiilor pentru dată și oră între diferiții producători de sisteme DBMS. În mare parte, motivul este faptul că producătorii au dezvoltat propriile tipuri de date temporale înainte de stabilirea standardelor. Din cauza acestei diversități, funcțiile pentru dată și oră sunt prezentate pe scurt pentru Microsoft SQL Server, Oracle, DB2 și MySQL. Termenii scriși cu caractere italice sunt definiți în notele care urmează după fiecare tabel. În toată această secțiune, termenul *data_și_ora* înseamnă un șir de caractere care conține informații despre dată și oră, într-un format acceptabil pentru sistemul DBMS respectiv.

Funcții pentru dată și oră în Microsoft SQL Server

Microsoft SQL Server oferă funcțiile pentru dată și oră prezentate în tabelul următor:

Funcție	Scop	Parametri de intrare
DATEADD	Returnează o nouă valoare *data_și_ora*, adăugând un interval la valoarea *parte_de_dată* furnizată ca parametru	*parte_de_dată*, interval, *data_și_ora*
DATEDIFF	Returnează numărul de intervale *data_și_ora* trecute între două date	*parte_de_dată*, dată de început, dată de sfârșit
DATENAME	Returnează un nume sub formă de text, reprezentând *partea_de_dată* selectată pentru valoarea *data_și_ora* furnizată ca intrare	*parte_de_dată*, *data_și_ora*

Funcție	Scop	Parametri de intrare
DATEPART	Returnează o valoare întreagă, reprezentând *partea_de_dată* selectată pentru valoarea *data_și_ora* furnizată ca intrare	*parte_de_dată*, *data_și_ora*
DAY	Returnează o valoare întreagă, reprezentând ziua conținută în valoare a *data_și_ora* furnizată ca intrare	*data_și_ora*
GETDATE	Returnează valoarea curentă *data_și_ora* furnizată de sistem	Nu are
GETUTCDATE	Returnează valoarea curentă *data_și_ora* UTC (Universal Coordinated Time)	Nu are
MONTH	Returnează o valoare întreagă, reprezentând luna conținută în valoare a *data_și_ora* furnizată ca intrare	*data_și_ora*
YEAR	Returnează o valoare întreagă, reprezentând anul conținut în valoarea *data_și_ora* furnizată ca intrare	*data_și_ora*

NOTĂ: Valoarea parte_de_dată *este un parametru care specifică o parte a datei, cum ar fi anul, luna, ziua, ora, minutul, secunda și milisecunda. Consultați documentația Microsoft SQL Server pentru informații despre valori și opțiuni.*

Funcții pentru dată și oră în Oracle

Oracle are mai multe de 24 de funcții pentru dată și oră. Rețineți că Oracle numește acest tip de date DATE și toate datele au și o componentă pentru oră, care are valoarea zero (reprezentând miezul nopții) atunci când nu este folosită. Funcțiile pe care este cel mai probabil să le folosiți sunt prezentate în tabelul următor:

Funcție	Scop	Parametri de intrare
ADD_MONTHS	Adaugă numărul de luni specificat la data furnizată ca parametru	dată, numărul de luni (o valoare pozitivă sau negativă)
CURRENT_DATE	Returnează data curentă în zona orară stabilită pentru sesiunea bazei de date	Nu are

Funcție	Scop	Parametri de intrare
EXTRACT	Extrage câmpul *data_și_ora* specificat din data furnizată ca intrare	*cuvânt cheie pentru data_și_ora*, dată
LAST_DAY	Returnează data furnizată, cu ziua înlocuită cu ultima zi a lunii respective	dată
MONTHS_BETWEEN	Returnează numărul de luni (inclusiv partea fracționară) între două date furnizate ca intrare; rezultatul este negativ dacă a doua dată este înaintea primei date	prima dată, a doua dată
SYSDATE	Returnează valoarea curentă *data_și_ora* furnizată de sistem	Nu are
TO_CHAR	Atunci când este folosită pentru o dată, funcția returnează un șir de caractere, în formatul specificat de *șirul_pentru_formatare*	dată, *șir_pentru_formatare*
TO_DATE	Transformă șirul de caractere furnizat ca intrare într-o dată formatată intern de Oracle, folosind *șirul_pentru_formatare* ca tipar pentru a interpreta conținutul șirului de caractere	dată, *șir_pentru_formatare*
TRUNC	Trunchiază o dată la unitatea de timp specificată de cuvântul cheie *data_și_ora*. Dacă acest parametru este omis, data este trunchiată la ziua curentă	dată, *cuvânt cheie pentru data_și_ora*

NOTĂ:

- *Cuvântul cheie pentru câmpul* data_și_ora *este un cuvânt cheie care specifică unul dintre câmpurile conținute de o dată Oracle, respectiv YEAR, MONTH, DAY, HOUR, MINUTE și SECOND.*
- Șir_pentru_formatare *este un șir de simboluri care specifică formatul folosit pentru dată atunci când aceasta este convertită în sau dintr-un șir de caractere. Există peste 40 de simboluri diferite care pot fi folosite în șirurile de formatare (puteți găsi lista completă în documentația Oracle). De exemplu, șirul de formatare 'MM/DD/YYYY HH:MI' are ca rezultat o dată, ca șir de caractere, de forma '12/01/2004 11:58', iar șirul de formatare 'DD-MON-RR' (formatul Oracle prestabilit) are ca rezultat o dată de forma '01-Dec-04'.*
- *Funcția TO_CHAR poate fi folosită și pentru transformarea valorilor numerice în șiruri de caractere.*

- *Funcția TRUNC poate fi folosită și pentru trunchierea valorilor numerice, având ca efect ștergerea cifrelor din dreapta punctului zecimal.*

Funcții pentru dată și oră în MySQL

MySQL are mai multe de 30 de funcții pentru dată și oră. Dintre acestea, cele pe care este cel mai probabil să le folosiți sunt prezentate în tabelul următor:

Funcție	Scop	Parametri de intrare
ADDDATE	Însumează două date, intervale sau expresii *data_și_ora*, returnând o nouă dată	expresie 1, expresie 2
ADDTIME	Însumează două expresii temporală, returnând o nouă valoare pentru oră	expresie 1, expresie 2
CURDATE	Returnează data curentă, în formatul YYYY-MM-DD (AAAA-LL-ZZ)	Nu are
DATE	Returnează partea de dată a unei date sau a unei expresii *data_și_ora*	expresie *data_și_ora*
DATEDIFF	Returnează numărul de zile dintre două date	dată de început, dată de sfârșit
DATE_FORMAT	Formatează o dată conform unui *șir_pentru_formatare*	dată, *șir_pentru_formatare*
DAYNAME	Returnează, ca nume de tip text, ziua din săptămâna conținută în dată	dată
DAYOFMONTH	Returnează ziua din lună, ca valoare de la 1 la 31	dată
DAYOFWEEK	Returnează numărul zilei din săptămână pentru o zi conținută de o dată (1 pentru duminică, 2 pentru luni și așa mai departe)	dată
DAYOFYEAR	Returnează numărul zilei din an pentru o zi conținută de o dată, ca valoare de la 1 la 366	dată
LAST_DAY	Schimbă ziua dintr-o dată cu ultima zi a lunii respective	dată
MONTH	Returnează numărul lunii dintr-o dată, ca valoare de la 1 la 12	dată
MONTHNAME	Returnează, ca nume de tip text, luna conținută în dată	dată
NOW	Returnează data și ora curente	Nu are

Funcție	Scop	Parametri de intrare
STR_TO_DATE	Transformă un șir de caractere într-o valoare *data_și_ora*; *șirul_pentru_formatare* indică formatul datei din șirul de caractere furnizat ca intrare	șir de caractere, *șir_pentru_formatare*
TIME	Extrage partea referitoare la oră dintr-o valoare *data_și_ora* sau dintr-o expresie temporală	*data_și_ora*
TIMEDIFF	Returnează diferența de timp dintre două valori *data_și_ora* sau două expresii temporale furnizate ca parametri	expresie 1, expresie 2
TIME_FORMAT	Formatează ora conform șirului de formatare	ora, *șir_pentru_formatare*
UTC_DATE	Returnează data UTC (Universal Coordinated Time) curentă	Nu are
UTC_TIME	Returnează ora UTC (Universal Coordinated Time) curentă	Nu are
WEEKOFYEAR	Returnează săptămâna din anul conținut într-o dată, ca valoare de la 1 la 54	dată

NOTĂ: Șirul_pentru_formatare *este un șir de caractere care specifică opțiunile de formatare pentru diferitele componente ale datei. Pentru detalii, consultați documentația MySQL.*

Funcții pentru dată și oră în DB2

DB2 are mai multe de 20 de funcții pentru dată și oră. Dintre acestea, cele pe care este cel mai probabil să le folosiți sunt prezentate în tabelul următor:

Funcție	Scop	Parametri de intrare
DATE	Transformă o expresie într-o dată	expresie
DAY	Returnează partea referitoare la zi dintr-o expresie *data_și_ora*	expresie *data_și_ora*
DAYNAME	Returnează numele, sub formă de text, al zilei din săptămână, dintr-o dată sau o expresie *data_și_ora*	expresie *data_și_ora*
DAYOFWEEK	Returnează numărul zilei din săptămână (1 pentru duminică, 2 pentru luni și așa mai departe) dintr-o expresie *data_și_ora*	expresie *data_și_ora*

Funcție	Scop	Parametri de intrare
DAYS	Returnează o valoare întreagă reprezentând o dată	expresie *data_și_ora*
MINUTE	Returnează partea referitoare la minute dintr-o expresie *data_și_ora*	expresie *data_și_ora*
MONTH	Returnează partea referitoare la lună dintr-o expresie *data_și_ora*	expresie *data_și_ora*
MONTHNAME	Returnează numele, sub formă de text, al lunii dintr-o dată sau o expresie *data_și_ora*	expresie *data_și_ora*
QUARTER	Returnează un întreg cu o valoare de la 1 la 4 reprezentând trimestrul din care face parte data furnizată ca parametru	expresie *data_și_ora*
SECOND	Returnează partea referitoare la secunde dintr-o expresie *data_și_ora*	expresie *data_și_ora*
TIME	Returnează partea referitoare la ora din zi dintr-o expresie *data_și_ora*	expresie *data_și_ora*
WEEK	Returnează ziua din an, ca valoare întreagă de la 1 la 54	expresie *data_și_ora*
YEAR	Returnează partea referitoare la an dintr-o expresie *data_și_ora*	expresie *data_și_ora*

Folosirea avantajelor oferite de vizualizări

Probabil vă amintiți din capitolul 1 că o *vizualizare (view)* este o interogare stocată în baza de date, care pune la dispoziția utilizatorului bazei de date un subset particularizat de date din unul sau mai multe tabele ale bazei de date. Rețineți că Microsoft Access folosește termenul *query (interogare)* în loc de vizualizare. Frumusețea vizualizărilor constă în faptul că, după ce sunt create, pot fi interogate ca și cum ar fi tabele reale. De fapt, nu este nevoie ca utilizatorul să știe dacă folosește o vizualizare sau un tabel real. Pot exista unele restricții la manipularea datelor (la inserare, actualizare și ștergere – consultați documentația sistemului DBMS pe care-l folosiți) dar interogările funcționează pentru vizualizări la fel ca și pentru tabele. Mai mult, vizualizările oferă și alte avantaje:

- Maschează coloanele pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează rândurile pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează operațiile complexe, precum uniunile
- Cresc performanțele interogărilor (în unele sisteme RDBMS, cum ar fi Microsoft SQL Server)

Exemplele din această secțiune prezintă vizualizări care vă oferă avantajele menționate mai sus. Vă amintiți din capitolul 4 că sintaxa generală pentru crearea unei vizualizări este

```
CREATE [OR REPLACE] VIEW nume_vizualizare AS interogare_sql;
```

Iată câteva exemple practice, pornind de la baza de date a magazinului de produse video:

- Reguli evidente de confidențialitate obligă administratorul magazinului să țină secrete datele de identificare ale angajaților (numerele de asigurări sociale) și salariile. Totuși, există un număr de aplicații bazate pe web care necesită furnizarea unor informații despre angajați, cum ar fi numele sau numele șefului lor (dacă există). În asemenea situații, aveți nevoie de o vizualizare care să poată fi folosită în siguranță de un dezvoltator, fără pericolul de a dezvălui în mod accidental informații confidențiale. Interogarea din vizualizare trebuie să realizeze o uniune (join) cu tabelul PERSON pentru a afla numele angajatului și (folosind o uniune externă) numele șefului acestuia. Mai mult, numele coloanelor trebuie să fie cât mai prietenoase pentru echipa de dezvoltare. Iată instrucțiunea SQL folosită pentru crearea vizualizării:

```
CREATE VIEW EMPLOYEE_LIST AS
 SELECT A.PERSON_ID AS ID,
        B.PERSON_GIVEN_NAME AS FIRST_NAME,
        B.PERSON_MIDDLE_NAME AS MIDDLE_NAME,
        B.PERSON_FAMILY_NAME AS LAST_NAME,
        C.PERSON_GIVEN_NAME AS MANAGER_FIRST_NAME,
        C.PERSON_FAMILY_NAME AS MANAGER_LAST_NAME
   FROM EMPLOYEE A JOIN PERSON B
             ON A.PERSON_ID = B.PERSON_ID
           LEFT OUTER JOIN PERSON C
             ON A.SUPERVISOR_PERSON_ID = C.PERSON_ID

View created.
```

După crearea vizualizării, puteți să o interogați ca și cum ar fi un tabel. Iată o interogare simplă făcută în vizualizarea pe care tocmai am creat-o:

```
SELECT LAST_NAME, MANAGER_LAST_NAME
  FROM EMPLOYEE_LIST
 ORDER BY LAST_NAME;

LAST_NAME                 MANAGER_LAST_NAME
--------------------      -----------------
Alexander
Bernstein                 Alexander
Chung                     Alexander
```

- Pentru crearea unui catalog cu filmele disponibile, aveți nevoie de o listă cu descrierile categoriilor MPAA și cu descrierile genurilor. Pare evident că veți avea nevoie foarte des de această uniune de trei tabele, așa că puteți ajuta personalul magazinului prin crearea unei vizualizări care conține deja uniunile necesare. Iată instrucțiunea SQL folosită pentru crearea vizualizării:

```
CREATE VIEW MOVIE_LISTING AS
SELECT A.MOVIE_ID, B.MOVIE_GENRE_DESCRIPTION AS GENRE,
       C.MPAA_RATING_CODE AS RATING,
       C.MPAA_RATING_DESCRIPTION AS RATING_DESC,
       A.MOVIE_TITLE, A.RETAIL_PRICE_VHS,
       A.RETAIL_PRICE_DVD, A.YEAR_PRODUCED
  FROM MOVIE A JOIN MOVIE_GENRE B ON
       A.MOVIE_GENRE_CODE = B.MOVIE_GENRE_CODE
              JOIN MPAA_RATING C ON
       A.MPAA_RATING_CODE = C.MPAA_RATING_CODE
```

Iată și o interogare simplă care folosește vizualizarea în locul tabelelor de bază:

```
SELECT GENRE, RATING, MOVIE_TITLE
  FROM MOVIE_LISTING
 ORDER BY GENRE, RATING, MOVIE_TITLE;

GENRE                RATING  MOVIE_TITLE
-------------------- ------  ------------------------------
Action and Adventure PG-13   Master and Commander: The Far
Action and Adventure PG-13   Pirates of the Caribbean: The
Action and Adventure PG-13   The Day After Tomorrow
Action and Adventure PG-13   The Italian Job
Action and Adventure R       Kill Bill: Vol. 1
Action and Adventure R       Man on Fire
Action and Adventure R       The Last Samurai
Comedy               PG-13   50 First Dates
Comedy               PG-13   Matchstick Men
Comedy               PG-13   Something's Gotta Give
Comedy               PG-13   The School of Rock
Drama                PG-13   Big Fish
Drama                R       Cold Mountain
Drama                R       Lost in Translation
Drama                R       Monster
Drama                R       Mystic River
Drama                R       Road to Perdition
Foreign              R       Das Boot
Romance              PG-13   13 Going on 30
Romance              PG-13   Two Weeks Notice

20 rows selected.
```

- Magazinul de produse video ar vrea să instaleze un chioșc informatizat în secțiunea de filme pentru copii, astfel încât tinerii să poată căuta filmele online. Totuși, administratorul magazinului dorește ca numai filmele din categoriile MPAA G, PG și PG-13 să fie disponibile în acest chioșc. Ar fi destul de simplu să impunem această regulă în aplicația care controlează chioșcul. Totuși, dacă ar fi folosită o vizualizare, iar aplicația din chioșcul respectiv ar fi obligată să folosească numai vizualizarea, nu ar putea apărea „scăpări" accidentale în logica aplicației și, ulterior, vizualizarea ar putea fi simplu ajustată, dacă MPAA ar modifica sistemul de clasificare (așa cum s-a mai întâmplat de câteva ori). În loc să repetați uniunea din vizualizarea MOVIE_LISTING, puteți folosi vizualizarea MOVIE_LISTING pentru a crea o nouă vizualizare. Da, puteți folosi o vizualizare pentru a crea o altă *vizualizare*! Acest mod de lucru vă oferă o flexibilitate inegalabilă. Iată un exemplu:

```
CREATE VIEW CHILDRENS_MOVIE_LISTING AS
SELECT * FROM MOVIE_LISTING
 WHERE RATING IN ('G','PG','PG-13')
```

Iată rezultatele obținute, folosind aceeași interogare ca și pentru vizualizarea MOVIE_LISTING:

```
SELECT GENRE, RATING, MOVIE_TITLE
  FROM CHILDRENS_MOVIE_LISTING
 ORDER BY GENRE, RATING, MOVIE_TITLE;

GENRE                RATING  MOVIE_TITLE
-------------------  ------  --------------------------------
Action and Adventure PG-13   Master and Commander: The Far
Action and Adventure PG-13   Pirates of the Caribbean: The
Action and Adventure PG-13   The Day After Tomorrow
Action and Adventure PG-13   The Italian Job
Comedy               PG-13   50 First Dates
Comedy               PG-13   Matchstick Men
Comedy               PG-13   Something's Gotta Give
Comedy               PG-13   The School of Rock
Drama                PG-13   Big Fish
Romance              PG-13   13 Going on 30
Romance              PG-13   Two Weeks Notice

11 rows selected.
```

Observați modul în care clauza WHERE din vizualizarea CHILDRENS_MOVIE_LISTING blochează rândurile care nu dorim să apară în rezultatele interogării. Atenție, totuși, la crearea vizualizărilor bazate pe alte vizualizări: vizualizarea CHILDRENS_MOVIE_LISTING se bazează pe vizualizarea MOVIE_LISTING și poate să devină invalidă sau să nu mai funcționeze corect dacă vizualizarea MOVIE_LISTING

este modificată. Fără o planificare atentă, întreaga construcție se va prăbuși ca un castel din cărți de joc de fiecare dată când faceți o modificare. Cu toate acestea, vizualizările sunt instrumente puternice, care nu pot fi ignorate.

Folosirea instrucțiunilor SQL pentru generarea altor instrucțiuni SQL

Majoritatea produselor RDBMS conțin o serie de vizualizări de tip catalog, care permit accesul prin interogări la metadatele ce descriu obiectele conținute în baza de date. Administratorii de baze de date experimentați știu cum să folosească datele din catalog într-un mod avantajos și pot scrie instrucțiuni SQL care generează alte instrucțiuni SQL.

Generarea instrucțiunilor SQL în Oracle

În Oracle, vizualizarea USER_TABLES conține informații despre toate tabelele care aparțin utilizatorului curent al bazei de date. Puteți să folosiți comanda „DESCRIBE USER_TABLES" pentru a vedea definiția acestei vizualizări sau să consultați manualul de referință al serverului Oracle pentru descrierea tuturor vizualizărilor de tip catalog.

Iată o instrucțiune SQL care creează o comandă DROP TABLE pentru toate tabelele din vizualizarea USER_TABLES. Clauza WHERE a fost adăugată pentru a elimina anumite tabele Oracle interne, care altfel ar putea apărea în setul de rezultate. Există tehnici pentru transmiterea rezultatelor interogării într-un fișier care poate fi apoi folosit ca script, dar acestea depășesc scopurile cărții de față (pentru detalii, vedeți comanda SQL*Plus SPOOL).

```
SELECT 'DROP TABLE ' || TABLE_NAME ||
       ' CASCADE CONTRAINTS;' AS SQL
  FROM USER_TABLES
 WHERE TABLE_NAME NOT LIKE 'BIN$%';

SQL
--------------------------------------------------
DROP TABLE MOVIE_RENTAL CASCADE CONSTRAINTS;
DROP TABLE CUSTOMER_TRANSACTION CASCADE CONSTRAINTS;
DROP TABLE CUSTOMER_ACCOUNT_PERSON CASCADE CONSTRAINTS;
DROP TABLE EMPLOYEE CASCADE CONSTRAINTS;
DROP TABLE MOVIE_LANGUAGE CASCADE CONSTRAINTS;
DROP TABLE MOVIE_COPY CASCADE CONSTRAINTS;
DROP TABLE MOVIE CASCADE CONSTRAINTS;
DROP TABLE PERSON CASCADE CONSTRAINTS;
DROP TABLE MPAA_RATING CASCADE CONSTRAINTS;
DROP TABLE MOVIE_GENRE CASCADE CONSTRAINTS;
DROP TABLE LANGUAGE CASCADE CONSTRAINTS;
DROP TABLE CUSTOMER_ACCOUNT CASCADE CONSTRAINTS;
```

Generarea instrucțiunilor SQL în Microsoft SQL Server

Microsoft folosește termenul „system tables" (tabele de sistem) pentru tabelele SQL Server care conține metadate. Puteți găsi descrierile acestor tabele în manualele Books Online, în secțiunea „system tables". Iată echivalentul în Microsoft SQL Server a instrucțiunii din exemplul precedent, folosind tabelul SQL Server SYSOBJECTS. Clauza WHERE filtrează toate obiectele, cu excepția celor de tip „tabel de utilizator".

```
SELECT 'DROP TABLE ' + NAME + ';'
  FROM SYSOBJECTS
 WHERE XTYPE='U'

DROP TABLE PERSON;
DROP TABLE MOVIE;
DROP TABLE MOVIE_COPY;
DROP TABLE MOVIE_LANGUAGE;
DROP TABLE EMPLOYEE;
DROP TABLE CUSTOMER_ACCOUNT_PERSON;
DROP TABLE CUSTOMER_TRANSACTION;
DROP TABLE MOVIE_RENTAL;
DROP TABLE CUSTOMER_ACCOUNT;
DROP TABLE LANGUAGE;
DROP TABLE MOVIE_GENRE;
DROP TABLE MPAA_RATING;
```

NOTĂ: Aceste instrucțiuni DROP generate nu vor funcționa decât cu unele modificări, din cauza restricțiilor referențiale. Microsoft SQL Server nu acceptă clauza CASCADE CONSTRAINTS pentru comanda DROP, așa că trebuie să eliminați restricțiile referențiale înainte de a putea șterge tabelele sau să aranjați instrucțiunile DROP în ordinea corespunzătoare înainte de a le putea rula.

Expresia CASE

Expresia CASE este o adăugire recentă, dar foarte importantă, la standardul SQL. Pentru prima dată, părți ale instrucțiunilor SQL pot fi executate condițional. De exemplu, o coloană din rezultatele interogării poate fi formatată în funcție de valorile conținute de o altă coloană. Totuși, este posibil ca implementarea SQL pe care să o folosiți să nu accepte încă această sintaxă, tocmai fiindcă este nouă.

Expresia CASE admite două forme generale.

Expresia CASE simplă

Iată sintaxa generală pentru forma simplă a expresiei CASE:

```
CASE expresie_de_intrare
   WHEN expresie_de_comparare THEN expresie_rezultat
  [WHEN expresie_de_comparare THEN expresie_rezultat...]
   ELSE expresie_rezultat
END
```

NOTĂ:

- *Fiecare condiție WHEN este evaluată sub forma* expresie_de_intrare = expresie_comparare *și, dacă rezultatul are valoarea TRUE logic, este returnată valoarea* expresie_rezultat *și nici o altă condiție WHEN nu mai este evaluată.*
- *Dacă nici una dintre condițiile WHEN nu este evaluată ca adevărată și există o condiție ELSE, este returnată valoarea* expresie_rezultat *asociată condiției ELSE.*
- *Dacă nici una dintre condițiile WHEN nu este evaluată ca adevărată și nu există o condiție ELSE, este returnată o valoare nulă.*

Ca exemplu, puteți folosi expresia CASE pentru a transforma valoarea MPAA_RATING_CODE într-un mesaj simplu, care poate fi afișat la terminalul de vânzare, pentru a aminti personalului magazinului să verifice vârsta clienților în cazul filmelor cu o categorie MPAA mai mare de PG-13. Remarcați plasarea cuvântului cheie AS imediat după cuvântul cheie END, pentru a asocia un nume coloanei generate în setul de rezultate. Iată exemplul:

```
SELECT MOVIE_ID, MPAA_RATING_CODE AS RATING,
  CASE MPAA_RATING_CODE
     WHEN 'G' THEN 'All ages'
     WHEN 'PG' THEN 'Parental guidance'
     WHEN 'PG-13' THEN 'Ages 13 and up'
     ELSE 'MUST be at least 17'
     END AS RATING_DESC
  FROM MOVIE
 ORDER BY MOVIE_ID;

MOVIE_ID  RATING   RATING_DESC
--------  ------   ------------------
       1  R        MUST be at least 17
       2  R        MUST be at least 17
       3  PG-13    Ages 13 and up
       4  PG-13    Ages 13 and up
       5  R        MUST be at least 17
       6  PG-13    Ages 13 and up
       7  PG-13    Ages 13 and up
       8  R        MUST be at least 17
       9  PG-13    Ages 13 and up
```

```
10  R       MUST be at least 17
11  PG-13   Ages 13 and up
12  PG-13   Ages 13 and up
13  PG-13   Ages 13 and up
14  R       MUST be at least 17
15  R       MUST be at least 17
16  PG-13   Ages 13 and up
17  PG-13   Ages 13 and up
18  R       MUST be at least 17
19  PG-13   Ages 13 and up
20  R       MUST be at least 17
```

Expresia CASE cu căutare

Așa-numita expresie CASE cu căutare (*searched CASE*) permite folosirea unor condiții de comparare mai flexibile, deoarece fiecare condiție este complet specificată, incluzând și operatorul de comparare. Iată sintaxa generală a expresiei:

```
CASE
   WHEN condiţie THEN expresie_rezultat
  [WHEN condiţie THEN expresie_rezultat...]
   ELSE expresie_rezultat
END
```

NOTĂ:

- *Fiecare* condiție *poate fi orice expresie SQL care este evaluată ca adevărată sau falsă.*
- *Condițiile WHEN sunt evaluate secvențial și, dacă una este evaluată ca adevărată, este returnată valoarea* expresie_rezultat *asociată și nici o altă condiție WHEN nu mai este evaluată.*
- *Dacă nici una dintre condițiile WHEN nu este evaluată ca adevărată și există o condiție ELSE, este returnată valoarea* expresie_rezultat *asociată condiției ELSE.*
- *Dacă nici una dintre condițiile WHEN nu este evaluată ca adevărată și nu există o condiție ELSE, este returnată o valoare nulă.*

Ca exemplu, iată o interogare care clasifică filmele în format VHS, în funcție de preț:

```
SELECT MOVIE_ID, RETAIL_PRICE_VHS,
   CASE
     WHEN RETAIL_PRICE_VHS IS NULL THEN 'Not Available'
     WHEN RETAIL_PRICE_VHS < 10 THEN 'Bargain'
     WHEN RETAIL_PRICE_VHS < 20 THEN 'Budget'
     WHEN RETAIL_PRICE_VHS < 40 THEN 'Average'
```

```
    ELSE 'Premium'
  END AS PRICE_CATEGORY
 FROM MOVIE
ORDER BY MOVIE_ID;

MOVIE_ID  RETAIL_PRICE_VHS  PRICE_CATEGORY
--------  ----------------  --------------
       1             58.97  Premium
       2             15.95  Budget
       3             14.95  Budget
       4             11.95  Budget
       5             24.99  Average
       6             24.99  Average
       7             14.95  Budget
       8             50.99  Premium
       9             12.98  Budget
      10             49.99  Premium
      11              6.93  Bargain
      12              9.95  Bargain
      13              6.93  Bargain
      14             24.99  Average
      15              9.99  Bargain
      16             11.69  Budget
      17             14.94  Budget
      18             24.99  Average
      19             12.98  Budget
      20             17.99  Budget
```

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Funcțiile SQL
   a. Returnează un set de valori
   b. Returnează o singură valoare
   c. Pot fi folosite în clauza WHERE a unei instrucțiuni SQL
   d. Pot fi folosite ca pseudonime pentru numele tabelelor într-o instrucțiune SQL
   e. Pot fi folosite în lista de coloane a unei instrucțiuni SQL

2. Funcția REPLACE
   a. Înlocuiește un nume de tabel cu un nume de vizualizare
   b. Înlocuiește numele unei coloane cu pseudonimul coloanei
   c. Înlocuiește un șir de caractere dintr-o coloană cu un alt șir de caractere
   d. Înlocuiește toate valorile dintr-o coloană cu un alt set de valori
   e. Înlocuiește toate rândurile dintr-o vizualizare cu rânduri conținând date dintr-un alt tabel
3. Funcția pentru valori nule
   a. Se numește NVL în bazele de date Oracle
   b. Se numește ISNULL în bazele de date IBM DB2
   c. Se numește ISNULL în bazele de date Microsoft SQL Server
   d. Se numește ISNULL în bazele de date MySQL
   e. Se numește IFNULL în standardul SQL
4. Funcția LTRIM
   a. Elimină spațiile de la sfârșitul șirurilor de caractere
   b. Elimină spațiile de la începutul șirurilor de caractere
   c. Poate fi imbricată cu alte funcții
   d. Înlocuiește valorile nule cu alte valori în șirurile de caractere
   e. Elimină spațiile de la începutul și de la sfârșitul șirurilor de caractere
5. Funcția CHAR
   a. Este numită CHR în unele implementări SQL
   b. Este identică cu funcția ASCII în unele implementări SQL
   c. Returnează valoarea corespunzătoare unui caracter din setul de caractere ASCII
   d. Returnează caracterul corespunzător unei valori din setul de caractere ASCII
   e. Transformă o valoare numerică într-un șir de caractere
6. Funcția SIGN
   a. Returnează -1 dacă parametrul furnizat are o valoare negativă
   b. Returnează 0 dacă parametrul furnizat are valoarea zero

c. Returnează +1 dacă parametrul furnizat are o valoare pozitivă
d. Returnează 0 dacă parametrul furnizat are valoarea NULL
e. Returnează o valoare nulă dacă parametrul furnizat nu este un număr

7. Funcția CEILING
   a. Rotunjește un număr prin adăugire până la primul număr întreg
   b. Rotunjește un număr prin scădere până la primul număr întreg
   c. Returnează întotdeauna un număr întreg
   d. Returnează un număr întreg sau o valoare nulă
   e. Este numită CEIL în unele implementări SQL
8. Funcția FLOOR
   a. Rotunjește un număr prin adăugire până la primul număr întreg
   b. Rotunjește un număr prin scădere până la primul număr întreg
   c. Returnează întotdeauna un număr întreg
   d. Returnează un număr întreg sau o valoare nulă
   e. Este numită FLR în unele implementări SQL
9. Funcțiile pentru dată și oră
   a. Sunt foarte asemănătoare între implementările diferiților producători
   b. Diferă foarte mult între implementările diferiților producători
   c. Includ funcții care formatează data și ora în vederea afișării
   d. Includ funcții care transformă șiruri de caractere în dată și oră
   e. Includ funcții care transformă data și ora în șiruri de caractere
10. Expresiile CASE
    a. Permit executarea condițională a clauzelor dintr-o instrucțiune SQL
    b. Există sub două forme, respectiv statice și dinamice
    c. Există sub două forme, respectiv cu căutare și fără căutare
    d. Există sub două forme, respectiv simple și cu căutare
    e. Există sub două forme, respectiv standard și cu căutare

Scrieți instrucțiunile SQL pentru următoarele probleme

11. Afișați valorile MPAA_RATING_CODE din tabelul MPAA_RATING, cu liniuțele de despărțire înlocuite cu spații.
12. Folosind funcția CHAR (numită CHR în Oracle), afișați valorile MOVIE_ID și MOVIE_TITLE pentru toate filmele care conțin apostrofuri (caracterul ASCII 39) în titlu.
13. Aflați valoarea ASCII a semnului de exclamare (!).
14. Aflați prețul mediu al filmelor în format DVD (coloana RETAIL_PRICE_DVD din tabelul MOVIE), cu media calculată rotunjită prin adăugire la cea mai apropiată sumă fără zecimale.
15. Aflați prețul mediu al filmelor în format VHS (coloana RETAIL_PRICE_VHS din tabelul MOVIE), cu media calculată rotunjită prin scădere la cea mai apropiată sumă fără zecimale.
16. Pentru o bază de date Oracle, generați comenzile SQL pentru eliminarea tuturor restricțiilor referențiale aflate în proprietatea utilizatorului curent. Vizualizarea Oracle de tip catalog se numește USER_CONSTRAINTS, iar restricțiile referențiale au valoarea 'R' în coloana CONSTRAINT_TYPE. Numele tabelului pe care se bazează restricția se află în coloana TABLE_NAME a vizualizării. Nu uitați că trebuie să folosiți comanda ALTER TABLE pentru eliminarea unei restricții.
17. Pentru o bază de date Microsoft SQL Server, afișați toate restricțiile de tip cheie externă. Folosiți tabelul de sistem SYSOBJECTS, în care NAME este numele restricției, iar coloana XTYPE are valoarea 'F' pentru restricțiile de tip cheie externă.
18. Scrieți o instrucțiune SQL care afișează fiecare cont de client (CUSTOMER_ACCOUNT_ID) cu un șir de caractere bazat pe valoarea coloanei CHILD_RENTAL_ALLOWED_INDIC, care să conțină textul „Child Rental OK" dacă valoarea coloanei este 'Y' și textul „NO CHILD RENTAL" dacă valoarea este 'N'. Sugestie: Cel mai bine funcționează în acest caz o expresie CASE simplă.
19. Scrieți o instrucțiune SQL care afișează fiecare film (MOVIE_ID) și o valoare pentru deceniu, în funcție de valoarea coloanei YEAR_PRODUCED ('80, '90, '00 sau necunoscut). În acest caz v-ar fi de folos o expresie CASE cu căutare.
20. Scrieți o instrucțiune SQL care afișează fiecare închiriere a unui film (tabelul MOVIE_RENTAL) cu valoarea LATE_OR_LOSS_FEE clasificată astfel: None (valoare nulă sau zero), Minor (valoare < 10) sau Major (valoare > 10).

CAPITOLUL 7

Întreținerea datelor folosind instrucțiuni DML

Până acum am discutat despre instrucțiunile DDL (Data Definition Language), folosite pentru crearea și modificarea obiectelor care stochează și organizează datele din baza de date, și despre instrucțiunile DQL (Data Query Language), folosite pentru selectarea datelor din baza de date. Acest capitol prezintă instrucțiunile DML (Data Manipulation Language), o parte a limbajului SQL folosită pentru întreținerea datelor stocate în tabelele relaționale ale bazei de date.

Limbajul DML este format din trei comenzi SQL:

- **INSERT** Adaugă noi rânduri într-un tabel al bazei de date
- **UPDATE** Actualizează rândurile existente într-un tabel al bazei de date
- **DELETE** Șterge rânduri dintr-un tabel al bazei de date

Rețineți că instrucțiunile DML individuale afectează datele dintr-un singur tabel. Este posibil ca într-o instrucțiune DML să referiți și o vizualizare care conține date din mai multe tabele (adică o vizualizare care conține o uniune de tabele), dar, în acest caz, instrucțiunea DML poate referi numai coloane dintr-un *singur* tabel al vizualizării. Cu alte cuvinte, atunci când o instrucțiune DML folosește o vizualizare, toate coloanele vizualizării referite în instrucțiunea DML trebuie să corespundă unor coloane dintr-un singur tabel fizic al bazei de date.

Este foarte probabil ca sistemul DBMS pe care-l folosiți să vă pună la dispoziție o formă oarecare de tranzacții, în care o serie de instrucțiuni DML sunt tratate unitar, ca un bloc de tip „totul sau nimic", adică un grup de instrucțiuni care trebuie să reușească toate sau să eșueze toate. Această caracteristică este foarte utilă atunci când instrucțiunile SQL sunt folosite în tranzacții comerciale și este prezentată detaliat în capitolul 9. Dacă încercați instrucțiunile din acest capitol (ceea ce vă recomand să faceți), puteți să ignorați tranzacțiile, exceptând cazul în care folosiți Oracle. În cazul bazelor de date Oracle, suportul pentru tranzacții este activat automat, în momentul în care vă conectați la baza de date. Aceasta înseamnă că nici una dintre modificările pe care le faceți nu va fi făcută permanent în baza de date decât în momentul în care vă deconectați de la baza de date printr-o metodă normală de ieșire sau lansați o comandă de finalizare (commit). În situații normale, aceasta nu este o problemă, dar dacă sistemul dumneavoastră sau baza de date se blochează (ceea ce înseamnă o ieșire anormală din baza de date), toate modificările făcute de la conectarea la baza de date sunt automat anulate. Mai mult, dacă aveți mai multe conexiuni simultane la baza de date, modificările făcute într-o conexiune (sesiune) nu sunt vizibile într-o altă conexiune decât după finalizarea tranzacției (commit). Dacă folosiți Oracle și vreți să schimbați comportamentul prestabilit al bazei de date, veți găsi în capitolul 9 detalii despre cum puteți face acest lucru.

Chiar dacă suportul pentru tranzacții nu este activat în mod explicit, sistemul DBMS va trata fiecare instrucțiune DML individuală ca pe o tranzacție. Aceasta înseamnă că, dacă apare o eroare la modificarea unui rând identificat de instrucțiunea DML, sistemul DML va anula întreaga instrucțiune, așa că aceasta nu va afecta nici un din rândurile tabelului. Cu alte cuvinte, sistemul DBMS nu va permite niciodată apariția inconsecvențelor în baza de date. De exemplu, dacă încerc să șterg toate rândurile din tabelul MOVIE și apare o eroare la ștergerea celui de-al treilea rând (după ce au fost deja șterse primele două rânduri), întreaga instrucțiune eșuează și sistemul DBMS reface (unii ar spune „ca prin magie") cele două rânduri șterse.

Instrucțiunile SQL din acest capitol presupun că sistemul DBMS pe care îl utilizați folosește formatul AAAA-LL-ZZ pentru datele calendaristice. Dacă baza de date cu care lucrați folosește un alt format, va trebui să modificați corespunzător exemplele din acest capitol pentru a le face să funcționeze corect în sistemul dumneavoastră. Pentru bazele de date Oracle, instrucțiunea de mai jos modifică formatul datelor calendaristice pentru sesiunea curentă (baza de date revine la formatul prestabilit de fiecare dată când vă reconectați la baza de date):

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';
```

Un alt aspect de care este important să țineți seama sunt restricțiile definite în tabelul referit de instrucțiunea DML. Sistemul DBMS nu va efectua în baza de date nici o modificare care încalcă una din restricții. La formarea instrucțiunilor DML, trebuie să țineți seama de următoarele aspecte referitoare la restricțiile tabelului modificat:

- **Restricții de tip cheie primară** Atunci când inserați un nou rând într-un tabel, cheia primară a noului rând trebuie să fie unică în întregul tabel. Când modificați valoarea unei chei primare (ceea ce se întâmplă rareori), noua valoare trebuie să fie unică în întregul tabel.
- **Restricții de unicitate** Ca și-n cazul cheilor primare, coloanele pe care a fost definită o restricție de unicitate trebuie să aibă valori unice în întregul tabel.
- **Restricții referențiale** Nu puteți insera sau actualiza valoarea unei chei externe decât dacă există deja rândul părinte corespondent care conține valoarea cheii în coloana cheii primare. În sens invers, nu puteți șterge un rând părinte dacă există rânduri subordonate care referă valoarea din rândul părinte, decât dacă restricția a fost definită cu opțiunea ON DELETE CASCADE. În general, inserările în tabele trebuie să fie făcute ierarhic (mai întâi rândurile părinte, apoi rândurile copii), iar ștergerile trebuie făcute în ordine inversă (copiii înaintea părinților).
- **Restricții NOT NULL** În cazul instrucțiunilor INSERT, trebuie să specificați valori pentru toate coloanele cu restricții NOT NULL. În cazul instrucțiunilor UPDATE, nu puteți înlocui valorile unei coloane cu valori nule dacă pe coloana respectivă este definită o restricție NOT NULL. Dacă instrucțiunea DML referă o vizualizare, nu o puteți folosi într-o instrucțiune INSERT dacă una dintre coloanele obligatorii ale tabelului (o coloană cu o restricție NOT NULL) lipsește din definiția vizualizării.
- **Restricții de verificare (CHECK)** O instrucțiune INSERT sau UPDATE nu poate stoca într-o coloană o valoare care încalcă o restricție CHECK definită pentru coloana respectivă.

Instrucțiunea INSERT

Instrucțiunea SQL INSERT este folosită pentru inserarea noilor rânduri de date în tabele. Instrucțiunea are două forme de bază: una în care valorile coloanelor sunt specificate chiar în instrucțiune și alta în care valorile sunt selectate dintr-un tabel sau o vizualizare, folosind o subinterogare. Să aruncăm o privire asupra celor două forme.

Inserarea unui singur rând de date folosind clauza VALUES

Instrucțiunea INSERT care folosește o clauză VALUES poate crea un singur rând la fiecare rulare, deoarece valorile pentru rândul de date respectiv sunt specificate chiar în instrucțiune. Iată sintaxa generală a instrucțiunii:

```
INSERT INTO nume_tabel_sau_vizualizare
   [(lista_de_coloane)]
 VALUES (lista_de_valori);
```

Rețineți următoarele:

- Lista de coloane este opțională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, trebuie specificată o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca *întotdeauna* să includeți lista de coloane, deoarece omiterea acesteia face ca instrucțiunea INSERT să fie dependentă de definiția tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar și opțională, probabil instrucțiunea INSERT va eșua la următoarea rulare.
- Dacă lista de coloane este specificată, lista de valori trebuie să conțină o valoare pentru fiecare coloană din listă, în aceeași ordine. Cu alte cuvinte, între lista de coloane și lista de valori trebuie să există o corespondență unu-la-unu. Orice coloană care lipsește din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în lista de valori pentru specificarea unei valori nule pentru o coloană.
- În Microsoft SQL Server și Sybase Adaptive Server nu puteți insera valori într-o coloană pentru care a fost definită proprietatea IDENTITY. Proprietatea IDENTITY este o metodă frecvent folosită pentru a atribui valori secvențiale unei chei primare, cum ar fi coloana MOVIE_ID din tabelul MOVIE. Dacă pentru o coloană a fost definită proprietatea IDENTITY, puteți să omiteți coloana respectivă din instrucțiunea INSERT, iar sistemul DBMS va atribui cheii primare următoarea valoare unică disponibilă. O altă soluție este să dezactivați restricția, folosind următoarea instrucțiune:

  ```
  SET identity_insert nume_tabel ON
  ```

Iată un exemplu care conține două instrucțiuni INSERT, una care creează un nou rând în tabelul MOVIE, cu o listă opțională de coloane din care este omisă coloana RETAIL_PRICE_VHS, și alta care creează un rând în tabelul MOVIE_COPY pentru același timp, dar fără lista opțională de coloane:

```
INSERT INTO MOVIE
   (MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE,
    MOVIE_TITLE, RETAIL_PRICE_DVD, YEAR_PRODUCED)
 VALUES (21, 'Drama', 'PG-13', 'Ray', 29.95, '2004');

INSERT INTO MOVIE_COPY
 VALUES (21, 1, '2005-04-01', null, 'V');
```

Pentru Microsoft SQL Server și Sybase Adaptive Server, dacă este definită proprietatea IDENTITY pentru tabelul MOVIE, puteți folosi următoarea variantă a primei instrucțiuni INSERT:

```
INSERT INTO MOVIE
   (MOVIE_GENRE_CODE, MPAA_RATING_CODE,
    MOVIE_TITLE, RETAIL_PRICE_DVD, YEAR_PRODUCED)
 VALUES ('Drama', 'PG-13', 'Ray', 29.95, '2004');
```

Inserări masive folosind o instrucțiune SELECT internă

Așa cum probabil ați observat, este nevoie de foarte mult cod pentru a insera în tabel un singur rând de date folosind o instrucțiune INSERT cu clauză VALUES. O altă soluție, care poate fi folosită pentru a insera rânduri multiple într-un tabel, este forma care folosește o instrucțiune SELECT internă. Această formă este utilă și pentru stabilirea următoarei valori disponibile pentru o cheie primară cu valori secvențiale, cum este coloana MOVIE_ID din tabelul MOVIE. De asemenea, poate fi folosită atunci când creați un tabel temporar pentru testare și vreți să-l populați cu toate datele dintr-un alt tabel. Sintaxa generală a instrucțiunii este

```
INSERT INTO nume_tabel_sau_vizualizare
   [(lista_de_coloane)]
 SELECT instrucţiune_select;
```

Rețineți următoarele:

- Lista de coloane este opțională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, instrucțiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca *întotdeauna* să includeți lista de coloane, deoarece omiterea acesteia face ca instrucțiunea INSERT să fie dependentă de definiția tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar și opțională, probabil instrucțiunea INSERT va eșua la următoarea rulare.

- Dacă lista de coloane este specificată, instrucțiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din listă, în aceeași ordine. Cu alte cuvinte, între lista de coloane și setul de rezultate al instrucțiunii SELECT trebuie să existe o corespondență unu-la-unu. Orice coloană care lipsește din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în instrucțiunea SELECT pentru specificarea unei valori nule pentru o coloană.

Ca exemplu, să presupunem că toate filmele sunt acum disponibile și în limba franceză. Tabelul MOVIE_LANGUAGE conține rânduri pentru limba franceză pentru o parte din filme, așa că acum vrem să le adăugăm numai pe cele care lipsesc. Am ales în mod intenționat un exemplu mai dificil, pentru vă arăta cât de sofisticată poate fi instrucțiunea SELECT internă. Iată cum arată instrucțiunea INSERT care va rezolva această problemă:

```
INSERT INTO MOVIE_LANGUAGE
   (MOVIE_ID, LANGUAGE_CODE)
  SELECT MOVIE_ID, 'fr'
    FROM MOVIE
   WHERE MOVIE_ID NOT IN
      (SELECT MOVIE_ID
         FROM MOVIE_LANGUAGE
        WHERE LANGUAGE_CODE = 'fr');
```

Exemplul următor inserează un nou rând în tabelul MOVIE, folosind instrucțiunea SELECT pentru a găsi valoarea maximă din coloana MOVIE_ID, pe care o incrementează cu o unitate, obținând astfel cheia primară a noului rând:

```
INSERT INTO MOVIE
  (MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE,
   MOVIE_TITLE, RETAIL_PRICE_DVD, YEAR_PRODUCED)
 SELECT MAX(MOVIE_ID)+1, 'Drama', 'PG-13',
        'The Terminal', 24.95, '2004'
   FROM MOVIE;
```

Instrucțiunea UPDATE

Instrucțiunea UPDATE este folosită pentru actualizarea datelor din coloanele unui tabel (sau ale unei vizualizări). Iată sintaxa generală a instrucțiunii UPDATE:

```
UPDATE nume_tabel_sau_vizualizare
    SET nume_coloană = expresie
       [, nume_coloană = expresie...]
  [WHERE condiție];
```

Rețineți următoarele:

- Clauza SET conține o listă cu una sau mai multe coloane, împreună cu o expresie care specifică noua valoare pentru fiecare coloană. În esență, aceasta este o listă de perechi nume-valoare, separate prin virgule, cu un operator de egalitate între fiecare nume și valoare.
- *Expresia* poate fi o constantă, un alt nume de coloană sau orice altă expresie pe care SQL o poate transforma într-o singură valoare, care va fi apoi atribuită coloanei respective.
- Clauza WHERE conține o expresie care limitează rândurile actualizate. Dacă această clauză este omisă, motorul SQL va încerca să actualizeze toate rândurile din tabel sau din vizualizare.

Iată câteva exemple:

- Filmul adăugat mai devreme în acest capitol (cu MOVIE_ID = 21) a fost introdus cu un preț incorect. Prețul pentru copia VHS ar trebui să fie 29.95, iar prețul pentru copia DVD ar trebui să fie 34.95. Instrucțiunea următoare actualizează prețurile, introducând valorile corecte. Observați că instrucțiunea actualizează două coloane din același rând al tabelului MOVIE.

```
UPDATE MOVIE
   SET RETAIL_PRICE_VHS = 29.95,
       RETAIL_PRICE_DVD = 34.95
 WHERE MOVIE_ID = 21;
```

- Proprietarul magazinului de produse video a decis să crească salariul tuturor funcționarilor (EMPLOYEE_JOB_CATEGORY = 'C') cu 8 procente. Această instrucțiune va actualiza o singură coloană de pe mai multe rânduri (toate rândurile pentru care corespunde categoria postului). Puteți folosi un calcul pentru a crește fiecare salariu cu 8 procente, înmulțind valoarea existentă cu 1.08 și rotunjind rezultatul la două cifre zecimale, cu ajutorul funcției ROUND. Iată această instrucțiune:

```
UPDATE EMPLOYEE
   SET EMPLOYEE_HOURLY_RATE =
       ROUND(EMPLOYEE_HOURLY_RATE * 1.08, 2)
 WHERE EMPLOYEE_JOB_CATEGORY = 'C';
```

Instrucțiunea DELETE

Instrucțiunea DELETE șterge unul sau mai multe rânduri dintr-un tabel. Instrucțiunea poate să folosească și o vizualizare, dar numai dacă aceasta se bazează pe un singur tabel (ceea ce înseamnă că instrucțiunile DELETE nu pot fi folosite pentru vizualizări care

conțin uniuni). În instrucțiunile DELETE nu sunt referite niciodată coloane, deoarece instrucțiunea șterge rânduri întregi de date, inclusiv toate valorile datelor (toate coloanele) din rândurile afectate. Dacă vreți să ștergeți o singură valoare din rândurile existente, folosiți instrucțiunea UPDATE pentru a înlocui valorile respective cu valori nule (presupunând că valorile nule sunt permise în acele coloane). Sintaxa generală a instrucțiunii DELETE este

```
DELETE FROM nume_tabel_sau_vizualizare
[WHERE condiţie];
```

Rețineți următoarele:

- Clauza WHERE este opțională. Totuși, este folosită aproape întotdeauna, deoarece o instrucțiune DELETE fără o clauză WHERE încearcă să șteargă *toate* rândurile din tabel – ceea ce, în majoritatea cazurilor, nu este rezultatul pe care vreți să-l obțineți.
- Atunci când este inclusă, clauza WHERE specifică rândurile care urmează să fie șterse. Orice rând pentru care condiția WHERE este evaluată ca adevărată este șters din tabel.
- Rețineți că nu puteți șterge rânduri dacă încălcați astfel o restricție referențială. În general, rândurile subordonate trebuie șterse înainte rândurilor părinte.

Iată câteva exemple:

- Ștergeți filmul adăugat mai devreme în acest capitol (cu MOVIE_ID = 21). Observați că trebuie să ștergeți mai întâi rândurile corespondente din tabelele MOVIE_COPY și MOVIE_LANGUAGE, deoarece acestea sunt rânduri „copii" ale rândului din tabelul MOVIE. Dacă încercați să ștergeți mai întâi rândul din tabelul MOVIE, sistemul DBMS va afișa un mesaj de eroare referitor la încălcarea unei restricții referențiale.

```
DELETE FROM MOVIE_COPY
 WHERE MOVIE_ID = 21;

DELETE FROM MOVIE_LANGUAGE
 WHERE MOVIE_ID = 21;

DELETE FROM MOVIE
 WHERE MOVIE_ID = 21;
```

- Ștergeți din tabelul MOVIE_LANGUAGE toate rândurile pentru limba spaniolă (LANGUAGE_CODE = 'es'). Rețineți că în multe implementări SQL se face diferențierea literelor mari de cele mici, caz în care valoarea codului de limbă trebuie specificată cu litere mici pentru a se potrivi cu datele din tabel.

```
DELETE FROM MOVIE_LANGUAGE
 WHERE LANGUAGE_CODE = 'es';
```

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Limbajul DML include următoarele comenzi SQL:
   a. INSERT
   b. REMOVE
   c. UPDATE
   d. SELECT
   e. DROP
2. O instrucțiune DML poate referi
   a. Coloane din mai multe tabele
   b. Coloane dintr-un singur tabel
   c. O vizualizare care conține coloane dintr-un singur tabel
   d. Coloane ale unei vizualizări bazate pe un singur tabel
   e. Coloane ale unei vizualizări bazate pe mai multe tabele
3. La formarea unei instrucțiuni DML, trebuie să țineți seama de următoarele restricții
   a. Restricții referențiale
   b. Restricții de securitate
   c. Restricții NOT NULL
   d. Restricții de unicitate
   e. Restricții cheie primară
4. O instrucțiune INSERT cu o clauză VALUES
   a. Trebuie să aibă o listă de coloane
   b. Trebuie să aibă o listă de valori
   c. Poate insera rânduri multiple la o singură rulare
   d. Poate folosi cuvântul cheie NULL pentru a atribui valori nule coloanelor
   e. Poate include o clauză WHERE

5. O instrucțiune INSERT cu o comandă SELECT imbricată este utilă pentru
   a. Găsirea următoarei valori pentru o cheie primară atribuită secvențial
   b. Mutarea rândurilor dintr-un tabel în altul
   c. Popularea unui tabel de test cu date dintr-un alt tabel
   d. Eliminarea rândurilor duplicate dintr-un tabel
   e. Inserarea mai multor rânduri într-un tabel cu o singură instrucțiune
6. O instrucțiune INSERT cu o comandă SELECT imbricată
   a. Trebuie să aibă două liste de coloane, una în clauza INSERT și una în instrucțiunea SELECT internă
   b. Trebuie să aibă o instrucțiune SELECT internă care returnează un singur rând de date
   c. Poate folosi cuvântul cheie NULL pentru a atribui valori nule coloanelor
   d. Poate include o clauză WHERE
   e. Trebuie să aibă în instrucțiunea SELECT internă o listă de coloane care corespunde cu lista de coloane din clauza INSERT sau cu coloanele din tabelul în care sunt inserate datele
7. O instrucțiune UPDATE fără o clauză WHERE
   a. Actualizează toate rândurile din tabel cu valori nule
   b. Încearcă să actualizeze toate rândurile din tabel
   c. Eșuează și returnează o eroare
   d. Șterge toate rândurile din tabel
   e. Are ca rezultat un produs cartezian
8. O instrucțiune DELETE fără o clauză WHERE
   a. Actualizează toate rândurile din tabel cu valori nule
   b. Eșuează și returnează o eroare
   c. Încearcă să șteargă toate rândurile din tabel
   d. Are ca rezultat un produs cartezian
   e. Trebuie să includă o listă de coloane

9. O instrucțiune UPDATE
   a. Trebuie să includă o clauză SET
   b. Trebuie să furnizeze o nouă valoare pentru cel puțin o coloană
   c. Trebuie să includă o clauză WHERE
   d. Poate atribui unei coloane valoarea unei alte coloane
   e. Poate atribui unei coloane o listă de valori derivate dintr-o expresie
10. O instrucțiune DELETE
   a. Poate include o listă opțională de coloane
   b. Poate include o clauză WHERE opțională
   c. Poate folosi cuvântul cheie FORCE pentru a forța ștergerea rândurilor
   d. Nu poate încălca restricțiile referențiale ale tabelului
   e. Poate avea o instrucțiune SELECT internă, ca parte a clauzei WHERE
11. Clauza SET dintr-o instrucțiune UPDATE poate atribui unei coloane o valoare care este
   a. O constantă
   b. Numele unei alte coloane
   c. O listă de valori
   d. Orice expresie din care rezultă o singură valoare
   e. Cuvântul cheie NULL
12. Instrucțiunea SELECT internă a unei instrucțiuni INSERT poate include
   a. O clauză WHERE
   b. O clauză GROUP BY
   c. Una sau mai multe funcții de agregare
   d. O uniune a mai multor tabele
   e. O clauză UNOIN

Scrieți instrucțiunile SQL pentru următoarele probleme

13. Folosind o instrucțiune INSERT cu o clauză VALUES, dar fără listă de coloane, inserați în tabelul MOVIE_GENRE un nou rând, cu valorile MOVIE_GENRE_CODE = 'TRAIN' și MOVIE_GENRE_DESCRIPTION = 'Training'.

14. Folosind o instrucțiune INSERT cu o clauză VALUES și o listă de coloane, inserați în tabelul MOVIE un nou rând, cu următoarele valori pentru date:

MOVIE_ID:	99
MOVIE_GENRE_CODE:	'TRAIN'
MPAA_RATING_CODE:	NR
MOVIE_TITLE:	Employee Training Video

15. Folosind o instrucțiune INSERT cu o instrucțiune SELECT internă, inserați în tabelul MOVIE_LANGUAGE un rând pentru limba japoneză (LANGUAGE_CODE = 'ja'), asociat cu toate filmele din tabelul MOVIE. (În prezent, tabelul MOVIE_LANGUAGE nu conține nici un rând pentru limba japoneză.)

16. Creați un tabel, numit RENTAL_TOTAL, folosind instrucțiunea CREATE de mai jos. Folosind o instrucțiune INSERT cu o instrucțiune SELECT internă, inserați în tabelul RENTAL_TOTAL un rând pentru fiecare film din tabelul MOVIE_RENTAL, conținând numărul total de închirieri și suma totală obținută din închirierea filmului respectiv.

```
CREATE TABLE RENTAL_TOTAL
   (MOVIE_ID           INTEGER        NOT NULL,
    NUMBER_OF_RENTALS  INTEGER        NOT NULL,
    TOTAL_RENTAL_FEES  NUMERIC(7,2)   NOT NULL);
```

17. Ștergeți toate rândurile din tabelul RENTAL_TOTAL.

18. Ștergeți rândurile pentru limba japoneză, create la întrebarea 15.

19. Copia 1 a filmului 1 (MOVIE_ID = 1, COPY_NUMBER = 1) a fost vândută pe data de 15 ianuarie 2005. actualizați rândul respectiv din tabelul MOVIE_COPY, actualizând în mod corespunzător coloana DATE_SOLD.

20. Actualizați tabelul MOVIE astfel încât să creșteți cu 10 procente toate prețurile pentru formatul VHS (RETAIL_PRICE_VHS) care nu conțin valori nule.

CAPITOLUL 8

Aplicarea controalelor de securitate folosind DCL

Securitatea a devenit un element vital în sistemele de calcul moderne. Nimic nu poate fi mai supărător pentru o organizație decât un articol într-un ziar referitor la date confidențiale sau secrete de fabricație furate electronic din sistemele de calcul. Legile federale recente din Statele Unite ale Americii privitoare la securitate impun restricții severe pentru utilizarea informațiilor personale. În California, o nouă lege impune ca această notificare să fie trimisă către orice locuitor rezident al statului California care ar putea datele compromise. Noile legi măresc semnificativ costul și

complicațiile necesare în efortul de păstrare a ordinii, și se pare că în viitor vor fi adoptate legi și mai exigente la nivel de stat și la nivel federal.

În acest capitol sunt prezentate o serie de concepte generale privitoare la securitate care trebuie luate în considerare în orice sistem de calculatoare. Accentul va fi pus totuși asupra modului în care este folosit limbajul SQL pentru a implementa controalele de securitate pentru accesul la date. Un administrator de baze de date bun se va implica în toate aspectele privind protejarea calculatorului și a rețelei în care este instalată baza de date, însă măsurile de securitate implementate în afara sistemului DBMS nu fac obiectul acestei cărți.

De ce este necesară securitatea?

Una dintre legile lui Murphy stabilește că orice poate merge rău va merge rău. Profesioniștii experimentați în securitatea sistemelor de tehnologia informațiilor vă vor spune însă că Murphy a fost un optimist. Rapoartele referitoare la breșele de securitate din sistemele de calcul sunt foarte răspândite. Există multe amenințări, plecând de la programe răuvoitoare (cum sunt virușii și modulele spion) și până la hackeri și hoți care vor să fure identități și orice lucru de valoare care poate fi obținut folosind o identitate sau o carte de credit furată. De fapt, furtul de identități și frauda cu cărți de credit se află în prezent pe primul loc într-un clasament al criminalității în Statele Unite (cele două infracțiuni sunt combinate într-o singură categorie pentru că foarte des una dintre ele conduce la cealaltă). Efectele negative ale breșelor de securitate variază de la simpla deranjare a utilizatorilor și până la pierderi de milioane de dolari. În unul dintre cele mai ciudate cazuri, hoții au furat identitatea unui cuplu de vârstnici și le-au vândut casa în timp ce victimele locuiau încă în ea – cuplul de vârstnici a aflat acest lucru abia după ce au primit un ordin de evacuare. Le-au fost necesare mai multe luni pentru a scăpa de aceste acuzații și a reface creditul inițial.

Un sistem de calcul este la fel de sigur cât este cea mai slabă componentă a sa. Din acest motiv, trebuie efectuată o abordare completă a securității și fiecare componentă trebuie făcută cât mai rezistentă posibil la intruziuni. Aceasta include protejarea rețelei de calculatoare, a serverelor în care rulează aplicațiile și bazele de date, a sistemelor de operare ale serverelor, a fișierelor de date care conțin informațiile bazei de date (inclusiv a copiilor de siguranță), a sistemului DBMS, a accesului la datele din DBMS și a sistemelor client care se conectează la servere și baze de date. Serverele cu configurații și parole prestabilite plasate în Internet au fost compromise în câteva *minute*. Parolele prestabilite ale bazelor de date și vulnerabilitățile uzuale de securitate sunt cunoscute pe scară largă. La începutul anului 2003, viermele Slammer a infectat zeci de mii de baze de date Microsoft SQL Server configurate cu un cont prestabilit "sa" (system administrator) fără parolă. În mod curios, cea mai mare pagubă provocată de acest vierme a fost pierderea serviciului atunci când calculatoarele infectate au trimis sute de mii de pachete în rețea în căutarea altor calculatoare pe care să le infecteze. Dacă dumneavoastră credeți că aceste lucruri nu vi se

pot întâmpla, mai gândiți-vă o dată. Iată câteva motive pentru care securitatea trebuie proiectată în sistemele de calcul:

- Bazele de date conectate la Internet sau la orice rețea sunt vulnerabile în fața hackerilor și a altor infractori care sunt hotărâți să strice sau să fure date. Între aceștia se numără:
  - Spioni ai concurenței, care încearcă să vă afle secretele.
  - Hackeri interesați să devină cunoscuți în urma penetrării sistemelor dumneavoastră.
  - Persoane individuale interesate să obțină lucruri valoroase.
  - Angajați nemulțumiți (pare ciudat că nu auzim niciodată de angajați mulțumiți, ci numai de angajați nemulțumiți).
  - Fanatici interesați să facă declarații politice pe socoteala organizației dumneavoastră.
  - Persoane instabile emoțional sau răuvoitoare.
- Încercări de fraudare. Orice auditor bancar vă va spune că 80% dintre fraude sunt comise de către sau cu ajutorul angajaților. Ca urmare, nu trebuie să presupuneți să sistemul dumneavoastră este imun doar pentru că baza de date nu este accesibilă din Internet.
- Greșeli neintenționate făcute de utilizatori autorizați care pot provoca breșe de securitate, pierderea datelor și erori de prelucrare.
- Controalele de securitate mențin utilizatorii onești la fel cum reușesc lacătele puse la case și birouri.

Trebuie înțeles faptul că măsurile de securitate nu pot împiedica în totalitate cel mai decis adversar să pătrundă într-un sistem. Singura modalitate sigură de a garanta că un sistem nu poate fi penetrat este de a-l opri și de a-l lăsa în această stare. Măsurile recomandate de precauție conduc însă la *încetinirea* chiar și a celui mai decis și mai priceput adversar suficient de mult cât să permită detectarea acestuia și intervenția. Mai presus de orice, utilizarea funcțiilor de securitate la toate nivelurile sistemelor oferă cea mai bună protecție pentru resursele valoroase de date.

Arhitecturi de securitate pentru o bază de date

Pentru administratorii de baze de date care folosesc baze de date de la mai mulți producători, una dintre provocări este că, exceptând Microsoft SQL Server și Sybase Adaptive Server, nu există două sisteme care să aibă aceeași arhitectură pentru securitatea bazei de date.

Desigur, acesta este efectul secundar al faptului că arhitecturile generale ale bazelor de date sunt diferite. Singurul motiv pentru care Microsoft SQL Server și Sybase Adaptive Server au arhitecturi similare este faptul că primul sistem a derivat din al doilea. Deoarece Microsoft SQL Server și Oracle sunt printre cele mai răspândite sisteme de baze de date în prezent, haideți să analizăm rapid modul în care este implementată securitatea bazei de date în fiecare dintre cele două sisteme.

Securitatea bazelor de date Microsoft SQL Server și Sybase Adaptive Server

În cazul sistemelor Microsoft SQL Server și Sybase Adaptive Server, după ce modulele software pentru DBMS sunt instalate în server, este creat un server de baze de date. Acesta este un termen care poate induce în eroare, desigur, pentru că de obicei denumim "server" sistemul hardware. În acest caz, *serverul* sau *serverul SQL* reprezintă o copie a modulelor software DBMS care rulează în memorie ca un set de procese (denumite și *servicii* în mediul Windows) cu informațiile de control asociate stocate într-o bază de date specială în serverul SQL. Voi folosi termenul *server SQL (SQL Server)* pentru a defini modulele software DBMS și termenul *server de baze de date* (*database server*) pentru a defini platforma hardware pe care rulează baza de date. În această arhitectură, fiecare server SQL gestionează mai multe baze de date, fiecare bază de date reprezentând o grupare logică de date stabilită de proiectantul bazei de date. Figura 8-1 prezintă o imagine

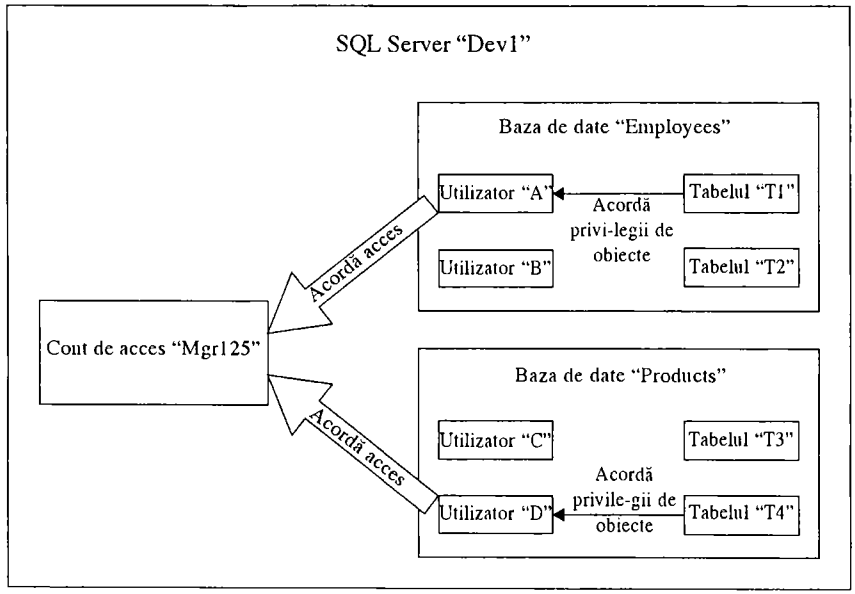


Figura 8-1 Securitatea bazei de date în Microsoft SQL Server și Sybase Adaptive Server

simplificată a arhitecturii de securitate pentru Microsoft SQL Server și Sybase Adaptive Server.

Securitatea în sistemele Microsoft SQL Server și Sybase Adaptive Server poate fi administrată fie prin instrumente grafice GUI (cum este Enterprise Manager), fie prin proceduri memorate furnizate de producător, apelate prin folosirea unor instrucțiuni SQL. În continuare este prezentată o listă de componente ale arhitecturii de securitate:

- **Contul de acces (login)** Acesta este un cont de utilizator din serverul SQL, denumit și *user login*. Este creat folosind un instrument GUI de genul Enterprise Manager sau procedura memorată sp_addlogin furnizată împreună cu sistemul DBMS. Contul de acces din serverul SQL nu este același lucru cu un cont de sistem de operare pe care utilizatorul îl poate avea în serverul de baze de date. În serverele de baze de date care rulează sistemul Microsoft Windows, contul de acces poate folosi autentificarea Windows, ceea ce înseamnă că sistemul de operare Windows stochează datele de identificare (numele de cont și parola) și autentifică utilizatorii atunci când efectuează conectarea la serverul SQL. Un avantaj evident al autentificării Windows este acela că accesul utilizatorului la diferitele servere SQL din organizație poate fi gestionat central prin contul Windows în loc să fie gestionat local de fiecare server SQL. Rețineți că după ce un cont de acces este definit în serverul SQL, utilizatorul bazei de date se poate conecta la serverul SQL, dar o simplă conectare nu îi oferă acces la nici o informație din baza de date. Există însă un cont de acces superior, denumit "sa" (system administrator) care, similar contului "root" din Unix și contului "Administrator" din Microsoft Windows, are drepturi depline asupra tuturor obiectelor din mediul SQL Server. Figura 8-1 prezintă un singur cont de acces de utilizator, denumit Mgr125.
- **Baza de date** O bază de date este o colecție logică de obiecte (tabele, vizualizări, indexuri și așa mai departe) definite de proiectantul bazei de date. Figura 8-1 prezintă două baze de date: Employees și Products. Este important să înțelegeți că unui cont de acces îi este permis să se conecteze la o bază de date numai după ce îi este atribuit acel privilegiu (drept) de către un administrator (studiați și subiectul „Utilizatorul", prezentat în continuare). Pe lângă bazele de date care păstrează datele de sistem, sunt create o serie de baze de date speciale o dată cu crearea serverului SQL (nu sunt prezentate în figura 8-1), care sunt utilizate de către sistemul DBMS pentru a gestiona serverul SQL. Între aceste baze de date se numără următoarele:
  - **master** Baza de date master conține informații la nivel de sistem, valori de configurare de inițializare, opțiuni de configurare, conturi de acces, lista bazelor de date configurate în serverul SQL și locația fișierelor principale de date ale bazelor de date.

  - **tempdb** Baza de date tempdb conține tabele temporare și proceduri memorate temporare.
  - **model** Baza de date model conține un șablon pentru toate celelalte baze de date create în sistem.
  - **msdb** Valabilă numai pentru bazele de date Microsoft SQL Server, baza de date msdb conține informații utilizate pentru planificarea acțiunilor și avertismentelor.
- **Utilizatorul** Fiecare bază de date are alocat un set de utilizatori. Fiecare utilizator al bazei de date este asociat unui cont de acces, astfel că fiecare utilizator este un pseudo-cont (adică un alias sau un pseudonim la un cont de acces SQL Server). Conturile de utilizator nu trebuie să aibă obligatoriu același nume de utilizator ca și conturile de acces corespondente. Atunci când un administrator alocă acces la o bază de date pentru un anumit cont de acces, utilizând fie un instrument GUI de genul Enterprise Manager, fie procedura memorată sp_addlogin furnizată împreună cu sistemul DBMS, contul de utilizator corespunzător contului de acces este creat de sistemul DBMS. În figura 8-1, contul Mgr125, corespunde utilizatorului A în baza de date Employees și utilizatorului D în baza de date Products. Aceste privilegii permit contului de acces să se conecteze la baza de date (sau la bazele de date) însă nu acordă utilizatorului nici un privilegiu asupra obiectelor din acele baze de date. Vom discuta modul de acordare a privilegiilor într-o altă secțiune din acest capitol.
- **Privilegiile** Fiecărui cont de utilizator dintr-o bază de date îi pot fi acordate oricâte drepturi (privilegii sau *permisiuni*). *Privilegiile de sistem* sunt privilegii generale aplicate la nivelul bazei de date. Microsoft SQL Server împarte aceste privilegii în *privilegii de server*, care includ drepturi referitoare la pornirea, oprirea și salvarea de siguranță a serverului SQL, și *privilegii de instrucțiuni*, care includ permisiuni cum ar fi crearea unei baze de date și crearea unui tabel. *Privilegiile de obiecte* permit acțiuni specifice asupra unui anumit obiect, cum ar fi permiterea operațiilor de selectare și actualizare în tabelul T1. Figura 8.1 conține săgeți care indică modul de acordare a privilegiilor de obiecte în tabelul T1 pentru utilizatorul A în baza de date Employees și, respectiv, în tabelul T4 pentru utilizatorul D în baza de date Products. Aceste privilegii funcționează aproape în același fel în toate bazele de date relaționale, datorită standardelor SQL; ca urmare, sunt prezentate în secțiunea "Privilegii la baza de date" din acest capitol.

Securitatea bazelor de date Oracle

Arhitectura de securitate Oracle, ilustrată în figura 8-2, este complet diferită în comparație cu arhitectura bazelor de date Microsoft SQL Server și Sybase Adaptive Server. Diferențele dintre cele două arhitecturi sunt evidențiate pe măsură ce este prezentată fiecare componentă:

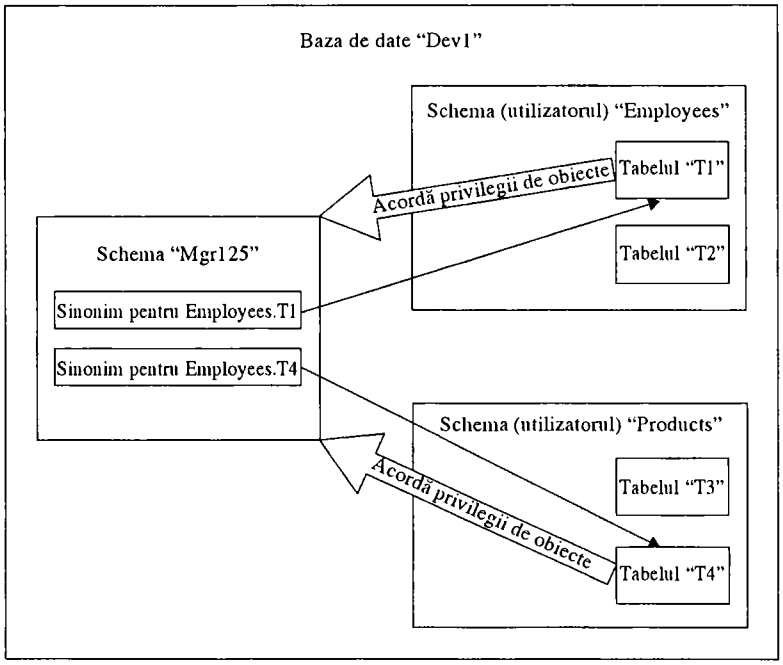


Figura 8-2 Securitatea bazei de date în Oracle

- **Instanța** Aceasta este o copie a modulului software DBMS Oracle care rulează în memorie. Fiecare instanță gestionează o singură bază de date.
- **Baza de date** Aceasta este o colecție de fișiere gestionate de o instanță Oracle unică. Privite împreună, instanța și baza de date Oracle formează ceea ce Microsoft SQL Server și Sybase Adaptive Server numesc *server SQL*. Figura 8-2 ilustrează baza de date Dev1.
- **Utilizatorul** Fiecare cont al bazei de date este numit *utilizator*. La fel ca în cazul Microsoft SQL Server și Sybase Adaptive Server, contul de utilizator poate fi autentificat extern (de către sistemul de operare) sau intern (de către DBMS). Fiecare utilizator are alocată o schemă (definită într-un paragraf ulterior), iar utilizatorul este *proprietarul* (*deținătorul*) schemei respective, ceea ce înseamnă că primește automat privilegii complete asupra oricărui obiect din schemă. La crearea bazei de date sunt creați automat următorii utilizatori predefiniți (nu sunt ilustrați în figura 8-2):
  - Utilizatorul SYS este proprietarul instanței Oracle, iar schema sa conține obiectele folosite de Oracle pentru a gestiona instanța. Acest utilizator este echivalent cu utilizatorul "sa" din Microsoft SQL Server și Sybase Adaptive Server.

  - Utilizatorul SYSTEM este proprietarul bazei de date Oracle, iar schema sa conține obiectele folosite de Oracle pentru a gestiona baza de date. Acest utilizator și schema sa corespunzătoare sunt similare utilizatorului "dbo" (database owner) din baza de date master în Microsoft SQL Server și Sybase Adaptive Server.
  - Multe opțiuni ale bazei de date Oracle creează conturi proprii de utilizatori atunci când sunt instalate.
- **Schema** Aceasta este colecția de obiecte din baza de date care aparțin unui anumit utilizator Oracle. Schema Oracle este echivalentă cu ceea ce Microsoft SQL Server și Sybase Adaptive Server denumesc *bază de date*. Figura 8-2 ilustrează schemele Employees, Products și Mgr125, care sunt deținute de utilizatorii Employees, Products și Mgr125. Numele schemelor și ale utilizatorilor sunt *întotdeauna* identice în Oracle. "Mgr125" este o soluție pentru o problemă specială pe care o veți întâlni în arhitectura de securitate Oracle și care va fi prezentată într-o altă secțiune din acest capitol, "Conturile deținătorilor de scheme".
- **Privilegiile** La fel ca în cazul sistemelor Microsoft SQL Server și Sybase Adaptive Server, privilegiile se împart în privilegii de sistem și privilegii de obiecte. Aceste privilegii sunt prezentate detaliat în secțiunea "Privilegiile bazei de date" din acest capitol.

Implementarea securității accesului la baza de date

Scopul securității accesului la baza de date este de a proteja datele de o eventuală utilizare neautorizată. Pentru a îndeplini acest obiectiv, administratorul bazei de date trebuie să stabilească utilizatorii care au permisiunea de a efectua anumite acțiuni cu anumite obiecte din baza de date. Permisiunile acordate utilizatorilor sunt numite *privilegii* ale bazei de date. În secțiunea următoare sunt analizate aceste privilegii, iar în continuare sunt prezentate instrucțiunile SQL utilizate pentru administrarea securității.

Privilegiile bazei de date

Privilegiile acordă utilizatorilor bazei de date autorizarea de a efectua diverse acțiuni în baza de date. Amintiți-vă că un utilizator al unei baze de date poate fi o persoană sau un proces care se conectează la baza de date. În general, atunci când unui utilizator al bazei de date i se acordă permisiunea de a se conecta la o bază de date, acesta nu poate să facă nimic fără a primi o serie de privilegii suplimentare. Altfel spus, un utilizator care se poate conecta la baza de date nu poate (în general vorbind) să acceseze nici o dată sau să efectueze

o funcție de administrare în baza de date până când nu i se acordă privilegiile pentru acțiunile respective. În SQL, privilegiile sunt acordate unui utilizator de bază de date prin instrucțiunea GRANT și sunt anulate prin instrucțiunea REVOKE; ambele instrucțiuni vor fi analizate detaliat în acest capitol. Privilegiile sunt împărțite în două categorii, descrise în secțiunile următoare: "Privilegiile de sistem" și "Privilegiile de obiecte".

Privilegiile de sistem

Privilegiile de sistem sunt permisiuni generale pentru efectuarea unor funcții de gestionare a serverului și a bazei (bazelor) de date. Fiecare producător de baze de date pune la dispoziție sute de permisiuni, majoritatea dintre ele fiind privilegii de sistem. În listele următoare sunt prezentate cele mai folosite privilegii. În documentațiile oferite de producătorul bazei de date sunt incluse detalii complete.

Cele mai folosite privilegii de sistem din Microsoft SQL Server sunt următoarele:

- **SHUTDOWN** Oferă posibilitatea de a executa comanda de oprire a serverului.
- **CREATE DATABASE** Oferă posibilitatea de a crea baze de date noi în serverul SQL.
- **BACKUP DATABASE** Oferă posibilitatea de a efectua salvări de siguranță ale bazelor de date din serverul SQL.

În continuare sunt prezentate cele mai folosite privilegii de sistem Oracle:

- **CREATE SESSION** Oferă posibilitatea de conectare la baza de date.
- **CREATE TABLE** Oferă posibilitatea de a crea tabele în schema proprie. Există privilegii similare și pentru alte tipuri de obiecte, cum ar fi indexuri, sinonime, proceduri și așa mai departe.
- **CREATE ANY TABLE** Oferă posibilitatea de a crea tabele în schema *oricărui* utilizator. Există privilegii similare și pentru alte tipuri de obiecte, cum ar fi indexuri, sinonime, proceduri și așa mai departe.
- **CREATE USER** Oferă posibilitatea de a crea utilizatori noi în baza de date.

Oracle permite includerea clauzei WITH ADMIN OPTION la acordarea permisiunilor de sistem. În cazul includerii acestei clauze, utilizatorul (sau utilizatorii) primesc privilegiul, dar dobândesc și capabilitatea de a acorda permisiuni altor utilizatori. *Nu* recomand această practică, pentru că deschide prea multe riscuri potențiale de securitate, în special datorită faptului că revocarea permisiunilor acordate în acest mod nu este derulată în cascadă.

Privilegiile de obiecte

Privilegiile de obiecte acordă autorizarea de a efectua anumite acțiuni specifice asupra anumitor obiecte din baza de date. De exemplu, un utilizator al bazei de date poate primi

privilegiile SELECT și UPDATE pentru un anumit tabel, dar nu neapărat și privilegiile INSERT sau DELETE. Instrucțiunea GRANT, care acordă privilegii la baza de date unui utilizator al bazei de date, poate include o clauză WITH GRANT OPTION care permite beneficiarului să acorde privilegiul respectiv altor utilizatori. Dacă privilegiul este revocat, are loc o revocare în cascadă în cazul în care utilizatorul a acordat permisiunea și altor utilizatori. *Nu* recomand utilizarea clauzei WITH GRANT OPTION pentru că este mult prea ușor să fie pierdut controlul asupra cui are privilegii și ce privilegii.

Instrucțiuni SQL folosite pentru administrarea securității

Această secțiune descrie instrucțiunile SQL folosite pentru a administra securitatea accesului la date. Rețineți că modalitățile în care este proiectată securitatea în diferite implementări SQL diferă foarte mult, așa că este recomandat ca întotdeauna să consultați documentația oferită de producătorului bazei de date.

Instrucțiunea CREATE USER

Mulți producători de baze de date pun la dispoziție o interfață grafică (GUI) pentru administrarea securității. Standardul SQL conține o instrucțiune CREATE USER ce poate fi folosită pentru a defini utilizatorii în baza de date. Sintaxa generală a instrucțiunii CREATE USER este următoarea:

```
CREATE USER nume_utilizator
   [IDENTIFIED BY parolă]
   [opțiuni];
```

NOTĂ:

- *Clauza IDENTIFIED BY este folosită pentru utilizatorii autentificați de sistemul DBMS. Pentru utilizatorii autentificați în exterior (de către sistemul de operare), sintaxa va fi diferită.*
- *Opțiunile (dacă sunt disponibile) sunt specifice sistemului DBMS și pot include valori predefinite pentru diferite proprietăți ale conexiunii la baza de date.*
- *Microsoft SQL Server și Sybase Adaptive Server nu acceptă instrucțiunea CREATE USER. Aceste sisteme folosesc proceduri memorate puse la dispoziție de producătorul bazei de date: sp_addlogin este folosită pentru a adăuga conturi de acces la serverul SQL, care oferă utilizatorilor capabilitatea de a se conecta la serverul SQL, iar sp_adduser este folosită pentru a adăuga utilizatori la o bază de date, care asociază un cont de acces cu un cont de utilizator pentru a permite accesul la o anumită bază de date.*

Instrucțiunea GRANT

Instrucțiunea GRANT este folosită pentru a acorda unul sau mai multe privilegii unui utilizator al bazei de date. Sintaxa generală a instrucțiunii GRANT este următoarea:

```
GRANT privilegiu [,privilegiu ...]
   [ON obiect]
    TO beneficiar [,beneficiar...]
   [WITH GRANT OPTION | WITH ADMIN OPTION];
```

NOTĂ:

- *Lista de privilegii poate conține unul sau mai multe privilegii de sistem ori unul sau mai multe privilegii de obiecte. Privilegiile de sistem și privilegiile de obiecte nu pot fi combinate în aceeași instrucțiune GRANT.*
- *Clauza ON* obiect *se aplică numai privilegiilor de obiecte. Clauza specifică obiectul pentru care sunt acordate privilegiile de obiecte din listă.*
- *Lista de beneficiari specifică unul sau mai mulți utilizatori ai bazei de date (sau roluri) care trebuie să primească privilegiile. Despre roluri vom discuta într-o altă secțiune din acest capitol.*
- *Clauza WITH GRANT OPTION sau WITH ADMIN OPTION permite beneficiarului de privilegiu să acorde la rândul său privilegiul altor utilizatori. Sintaxa și modul de folosire vor varia în funcție de implementarea SQL. După cum am menționat deja, nu recomand folosirea acestor clauze pentru că se pierde controlul asupra privilegiilor bazei de date.*

Urmează un exemplu. Veți găsi mai multe exemple în secțiunea "Simplificarea administrării cu ajutorul rolurilor" prezentată către finalul acestui capitol.

- Acordați utilizatorului PUBLIC privilegiul SELECT asupra tabelului MPAA_RATING. În majoritatea implementărilor SQL, PUBLIC este un cuvânt cheie folosit pentru a atribui un privilegiu tuturor utilizatorilor bazei de date. În general, nu este o idee prea bună, însă MPAA_RATING nu este decât un tabel de referință pentru coduri, așa că acordarea pentru toți utilizatorii a dreptului de a selecta acest tabel nu reprezintă un risc prea mare de securitate.

  ```
  GRANT SELECT ON MPAA_RATING TO PUBLIC
  ```

Instrucțiunea REVOKE

Sintaxa generală a instrucțiunii REVOKE este prezentată în continuare, alături de câteva exemple:

```
REVOKE privilegiu [,privilegiu ...]
   [ON obiect]
    FROM beneficiar [,beneficiar...];
```

NOTĂ:

- *Lista de privilegii poate conține unul sau mai multe privilegii de sistem ori unul sau mai multe privilegii de obiecte. Privilegiile de sistem și privilegiile de obiecte nu pot fi combinate în aceeași instrucțiune REVOKE.*
- *Clauza ON* obiect *se aplică numai privilegiilor de obiecte. Clauza specifică obiectul pentru care sunt revocate privilegiile de obiecte din listă.*
- *Lista de beneficieri specifică unul sau mai mulți utilizatori ai bazei de date (sau roluri) care urmează să piardă privilegiile. Despre roluri vom discuta într-o altă secțiune din acest capitol.*

În continuare este prezentat un exemplu – acesta anulează efectele instrucțiunii GRANT din secțiunea anterioară:

```
REVOKE SELECT ON MPAA_RATING FROM PUBLIC;
```

Conturile deținătorilor de scheme

În cazul tuturor bazelor de date, este recomandat să nu acordați utilizatorilor mai multe privilegii decât au nevoie pentru a-și îndeplini sarcinile. Această regulă previne apariția unor dezastre cu datele cauzate de erori umane (inclusiv erorile din programele de aplicații și din interogările la baza de date), dar are și rolul de a menține onestitatea oamenilor.

În Microsoft SQL Server și Sybase Adaptive Server, este recomandat să nu aveți utilizatori care se conectează la baza de date folosind contul "sa". Este bine să creați conturi de acces la baza de date care au privilegiile minimale necesare. Din nefericire, deseori această recomandare nu este aplicată, iar aplicațiile se conectează cu contul „sa" sau la o bază de date cu un cont de utilizator având rolul DBO (deținător al bazei de date) sau DBA (administrator al bazei de date). Rolurile sunt colecții de privilegii și vor fi prezentate în una dintre secțiunile următoare ale capitolului. Indiferent dacă este folosită din necunoaștere sau din lene, această practică reprezintă o amenințare gravă de securitate și ar trebui interzisă prin politica de securitate a organizației.

În figura 8-2, observați că utilizatorul Mgr125 nu deține nici un tabel, dar are anumite privilegii acordate de utilizatorii Employees și Products. Aceasta este o soluție pentru evitarea unei probleme grave existente în arhitectura de securitate Oracle. Dacă ați permis unui utilizator al bazei de date să se conecteze la baza de date folosind un utilizator de genul Employees sau Products, utilizatorul ar putea primi automat drepturi depline asupra tuturor obiectelor din schemă, inclusiv drepturile de introducere, ștergere și actualizare în orice tabel, precum și permisiunea de a crea și de a modifica tabele fără nici o restricție. În principiu, este același lucru cu permiterea folosirii utilizatorului „sa" sau a rolurilor DBO și DBA în Microsoft SQL Server și Sybase Adaptive Server. Utilizatorul Mgr125 simulează comportamentul contului de acces cu același nume ilustrat în figura 8-1. Acordând privilegiile corespunzătoare de sistem, puteți împiedica utilizatorul Mgr125 din Oracle să poată crea tabele proprii.

Ați observat probabil sinonimele pentru utilizatorul Mgr125 folosite în figura 8-2. Un *sinonim* este pur și simplu un pseudonim sau o poreclă pentru un obiect din baza de date. Sinonimele sunt folosite pentru comoditatea utilizatorului, astfel încât numele să nu fie calificate cu numele din schemă. Pentru a selecta direct obiecte din tabelul T1 în schema Employees, utilizatorul Mgr125 ar trebui să refere numele tabelului „Employees.T1" în instrucțiunea SQL. Acest mod de lucru nu este doar incomod, dar poate să provoace probleme în lanț dacă decideți să schimbați numele utilizatorului Employees. Prin crearea în schema Mgr125 a unui sinonim denumit „T1" care indică expresia Employees.T1, utilizatorul poate referi tabelul simplu „T1". Apropo, vă puteți aminti că toate numele de obiecte și de utilizatori din Oracle fac deosebirea între literele mari și literele mici, astfel că utilizarea în acest exemplu a unui amestec de litere este doar pentru demonstrație. Sintaxa pentru crearea acestui sinonim este următoarea:

```
CREATE SYNONYM T1 FOR EMPLOYEES.T1;
```

Simplificarea administrării folosind roluri

Un *rol* este o colecție numită de privilegii care pot, la rândul lor, să fie acordate unuia sau mai multor utilizatori. Majoritatea sistemelor RDBMS au roluri predefinite livrate împreună cu sistemul, iar utilizatorii din baza de date care au privilegiul CREATE ROLE pot crea roluri proprii. Rolurile au următoarele avantaje:

- **Rolurile pot exista înaintea conturilor de utilizator.** De exemplu, puteți să creați un rol care conține toate privilegiile necesare pentru a lucra într-un anumit proiect de dezvoltare. Atunci când un angajat nou este încadrat în echipa proiectului, o singură instrucțiune GRANT atribuie noului cont de utilizator toate permisiunile necesare.
- **Rolurile simplifică activitatea administratorului.** Atunci când este folosit un rol, pot fi acordate mai multe privilegii printr-o singură comandă.
- **Rolurile sunt menținute atunci când sunt eliminate conturile de utilizator.** În situațiile în care administratorul bazei de date trebuie să elimine și să re-creeze un cont de utilizator, ar putea fi necesară multă muncă pentru reintroducerea tuturor privilegiilor; această activitate este simplificată dacă toate privilegiile sunt introduse într-un rol.

Singurul dezavantaj potențial al rolurilor, în special al celor predefinite, este că pot fi acordate fără suficientă atenție asupra tuturor privilegiilor conținute, utilizatorul primind astfel mai multe privilegii decât minimum necesar. De exemplu, rolul CONNECT din Oracle include privilegiile CREATE SESSION și ALTER SESSION, după cum vă așteptați, dar mai include privilegiile CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE și CREATE VIEW. Aceasta este

probabil o colecție mai puternică de drepturi decât ar avea nevoie un utilizator obișnuit al bazei de date, așa că ar fi mai bine să primească doar privilegiul CREATE SESSION.

Pentru administratori, un rol obișnuit este DBA, care transmite o mulțime de privilegii puternice (peste 125 de privilegii individuale în Oracle). Evident, un astfel de rol foarte puternic trebuie acordat judicios.

Administrarea rolurilor în Microsoft SQL Server și Sybase Adaptive Server

Multe implementări SQL folosesc instrucțiunea SQL standard CREATE ROLE pentru a stabili roluri și instrucțiunea GRANT pentru a atribui unui utilizator privilegiile conținute într-un rol. Sistemele Microsoft SQL Server și Sybase Adaptive Server au însă o soluție proprie care necesită folosirea unor proceduri memorate furnizate de producător pentru activarea securității bazate pe roluri.

Rolurile sunt create folosind procedura memorată sp_addrole. Procedura memorată sp_droprole poate fi folosită pentru ștergerea rolului. Sintaxa generală pentru procedura memorată este următoarea:

```
sp_addrole 'nume_rol' [, 'nume_deținător']
```

- Parametrul *nume_rol* atribuie un nume rolului; numele trebuie să fie unic între numele de roluri și de utilizatori din baza de date curentă.
- Parametrul opțional *nume_deținător* specifică deținătorul rolului, valoarea prestabilită fiind „dbo" (contul de utilizator care deține baza de date).

În continuare sunt prezentate două instrucțiuni care adaugă roluri pentru directorul magazinului de produse video și pentru vânzătorii din magazin:

```
sp_addrole 'manager'
sp_addrole 'clerk'
```

Privilegiile sunt acordate rolului folosind o instrucțiune GRANT standard. În exemplele următoare, observați cum pot fi acordate aceleași privilegii mai multor roluri (în cazul în care cerințele organizației impun aceasta), însă, în același timp, și cum pot fi acordate privilegii diferite.

```
GRANT SELECT ON MPAA_RATING
   TO manager, clerk;
GRANT SELECT ON MOVIE
   TO clerk;
GRANT SELECT, INSERT, UPDATE, DELETE ON MOVIE
   TO manager;
```

Rolurile sunt alocate utilizatorilor bazei de date cu ajutorul procedurii memorate sp_addrolemember (furnizată de producător). Procedura memorată sp_droprolemember poate fi folosită pentru a anula unui utilizator un anumit rol. Sintaxa generală pentru procedura sp_addrolemember este următoarea:

```
sp_addrolemember 'nume_rol','nume_utilizator'
```

- Parametrul *nume_rol* specifică rolul care trebuie atribuit utilizatorului.
- Parametrul *nume_utilizator* specifică contul de utilizator din baza de date care va primi privilegiile conținute în rol.

Administrarea rolurilor în Oracle

Standardul SQL specifică utilizarea instrucțiunii CREATE ROLE pentru crearea rolurilor. Instrucțiunea DROP ROLE poate fi folosită ulterior pentru a șterge un rol (dacă este nevoie). Pentru a rula instrucțiunea, trebuie să aveți privilegiul CREATE ROLE, ceea ce de obicei înseamnă că trebuie să fiți un utilizator foarte puternic în Oracle (de exemplu, SYSTEM). Sintaxa generală a instrucțiunii CREATE ROLE este următoarea:

```
CREATE ROLE nume_rol [opțiuni];
```

- *nume_rol* specifică numele noului rol.
- *opțiuni* indică specificațiile opționale de securitate pentru noul rol.

În continuare sunt prezentate două instrucțiuni care adaugă roluri pentru directorul magazinului de produse video și pentru vânzătorii din magazin:

```
CREATE ROLE manager;
CREATE ROLE clerk;
```

La fel ca în cazul sistemelor Microsoft SQL Server și Sybase Adaptive Server, privilegiile sunt acordate unui rol cu ajutorul instrucțiunii GRANT. În exemplele următoare, observați cum pot fi acordate aceleași privilegii mai multor roluri (în cazul în care cerințele organizației impun aceasta), însă, în același timp, și cum pot fi acordate privilegii diferite.

```
GRANT SELECT ON MPAA_RATING
   TO manager, clerk;
GRANT SELECT ON MOVIE
   TO clerk;
GRANT SELECT, INSERT, UPDATE, DELETE ON MOVIE
   TO manager
```

Rolurile sunt atribuite utilizatorilor bazei de date cu ajutorul instrucțiunii GRANT. Sintaxa este aceeași cu sintaxa generală a instrucțiunii GRANT prezentată anterior în acest capitol.

Utilizarea vizualizărilor pentru implementarea securității la nivel de coloană și la nivel de rând

Una dintre cel mai întâlnite probleme de securitate care trebuie rezolvată este modul în care se permite utilizatorilor bazei de date accesul la anumite rânduri și coloane dintr-un tabel, interzicându-le în același timp accesul la alte rânduri și coloane. Vizualizările reprezintă soluții excelente pentru aceste probleme. Iată câteva dintre avantajele folosirii vizualizărilor pentru îndeplinirea obiectivelor de securitate:

- **Coloanele de care nu are nevoie un utilizator al bazei de date pot fi omise din vizualizare.** Presupunând că utilizatorului i-a fost acordat accesul la vizualizare și nu la tabelul de bază, această metodă împiedică în totalitate utilizatorul să vadă informațiile din coloanele omise din vizualizare.
- **În vizualizare poate fi inclusă o clauză WHERE pentru limitarea numărului de rânduri returnate.** Ca o metodă pentru limitarea numărului de rânduri, pot fi incluse uniuni pentru corespondența cu alte tabele. De exemplu, vizualizarea poate limita tabelul Products numai la acele produse care au identificatorul Division ID corespunzător compartimentului în care lucrează angajatul.
- **Pot fi utilizate uniuni la tabele „de căutare" pentru a înlocui valorile de cod dintr-un tabel cu descrierile corespunzătoare.** Un tabel de căutare conține de obicei o listă cu valori de cod (de exemplu, codurile de departamente, codurile de tranzacții, codurile de stare) și descrierile acestora, și este folosit pentru a „căuta" descrierile corespunzătoare codurilor. Deși este un aspect minor, utilizatorii care încearcă să acceseze fără permisiune înregistrările din baza de date în încercările de fraudare întâmpină dificultăți mult mai mari dacă nu pot vedea codurile folosite pentru împărțirea tranzacțiilor pe categorii. În plus, pentru angajații care încearcă să își îndeplinească sarcinile cât mai bine este recomandabil să citească și să înțeleagă descrierile de cod și nu valorile de cod corespunzătoare.

Există și alte metode de îndeplinire a acestor obiective. Multe sisteme RDBMS moderne (între care se numără Oracle și Microsoft SQL Server) includ specificații pentru securitatea la nivel de coloană, iar un administrator al bazei de date poate să acorde accesul pentru fiecare coloană de tabel. Pentru restricțiile la nivel de rând, există o funcție denumită Virtual Private Database (disponibilă în sistemele Oracle începând de la versiunea *9i*) care poate fi folosită în acest scop. În final, există utilizatori care preferă să folosească proceduri memorate pentru toate tipurile de acces la baza de date și ca urmare folosesc module proprii de program pentru a controla accesul la baza de date.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Legislația recentă referitoare la securitate
   a. Impune utilizarea rolurilor în administrarea securității bazei de date
   b. Restricționează utilizarea datelor personale
   c. Impune administratorilor de baze de date să protejeze rețeaua de calculatoare
   d. Impune plasarea identităților furate în Internet
   e. Impune avertizarea fiecărei persoane individuale care ar putea avea datele compromise
2. Securitatea este necesară pentru că
   a. Persoanele oneste fac greșeli
   b. Controalele de securitate ale aplicațiilor nu sunt adecvate
   c. 80% dintre fraude sunt comise de hackeri din exterior
   d. Bazele de date conectate la Internet sunt vulnerabile
   e. Controalele de securitate mențin onestitatea persoanelor
3. Între intrușii care încearcă să penetreze sistemele conectate la Internet se numără
   a. Auditori bancari
   b. Spioni ai concurenței
   c. Autori de jurnale personale web
   d. Hackeri
   e. Angajați nemulțumiți
4. Componentele care trebuie protejate sunt
   a. Stațiile de lucru client
   b. Serverele
   c. Bazele de date
   d. Sistemele de operare
   e. Rețelele

5. În Microsoft SQL Server, un cont de acces (cont de utilizator)
   a. Poate să se conecteze la oricâte baze de date
   b. Primește automat privilegii de acces la baza de date
   c. Poate folosi autentificarea Windows
   d. Poate fi autentificat de Microsoft SQL Server
   e. Deține o schemă a bazei de date
6. În Microsoft SQL Server, o bază de date
   a. Este deținută de un cont de acces
   b. Poate avea alocați unul sau mai mulți utilizatori
   c. Poate conține date de sistem (de exemplu, master) și date de utilizator (aplicație)
   d. Poate avea atribuite privilegii
   e. Este o colecție logică de obiecte de baze de date
7. În Oracle, un cont de utilizator
   a. Se poate conecta la oricâte baze de date
   b. Primește automat privilegii la baza de date
   c. Poate folosi autentificarea sistemului de operare
   d. Poate fi autentificat de sistemul DBMS Oracle
   e. Deține o schemă a bazei de date
8. În Oracle, o bază de date
   a. Este deținută de un utilizator
   b. Poate avea definite unul sau mai multe conturi de utilizator
   c. Poate conține date de sistem (de exemplu, schema sistemului) și date de utilizator (aplicație)
   d. Este același lucru cu o schemă
   e. Este gestionată de o instanță Oracle
9. Privilegiile de sistem
   a. Sunt acordate într-o manieră similară în Oracle, Sybase și Microsoft SQL Server
   b. Sunt specifice unui obiect al bazei de date
   c. Permit beneficiarului să efectueze o serie de funcții administrative pe server, cum ar fi oprirea serverului
   d. Sunt anulate cu ajutorul instrucțiunii SQL REMOVE
   e. Variază între bazele de date provenite de la producători diferiți

10. Privilegiile de obiecte
    a. Sunt acordate într-o manieră similară în Oracle, Sybase și Microsoft SQL Server
    b. Sunt specifice unui obiect al bazei de date
    c. Permit beneficiarului să efectueze o serie de funcții administrative pe server, cum ar fi oprirea serverului
    d. Sunt anulate cu ajutorul instrucțiunii SQL REMOVE
    e. Sunt atribuite cu ajutorul instrucțiunii SQL GRANT
11. Utilizarea clauzei WITH GRANT OPTION la atribuirea privilegiilor de obiecte
    a. Permite beneficiarului să acorde privilegiul altor utilizatori
    b. Acordă privilegiile DBA ale beneficiarului întregii baze de date
    c. Poate conduce la probleme de securitate
    d. Va fi revocată în cascadă dacă privilegiul este ulterior revocat
    e. Este o procedură recomandată pe scară largă pentru că este comodă în utilizare
12. Rolurile
    a. Pot fi atribuite unui singur utilizator
    b. Pot fi partajate de mai mulți utilizatori
    c. Pot exista înainte de existența utilizatorilor
    d. Pot conține oricâte privilegii de obiecte
    e. Pot conține un singur privilegiu de obiecte
13. Posibilele dezavantaje ale utilizării rolurilor pentru securitate sunt
    a. Sunt mai dificil de administrat decât privilegiile individuale
    b. Sunt eliminate atunci când este șters utilizatorul
    c. Sunt eliminate atunci când sunt șterse privilegiile
    d. Pot fi atribuite fără a lua în considerare toate privilegiile conținute
    e. Este necesară o perioadă suplimentară de instruire pentru administratorii care trebuie să le folosească
14. Vizualizările pot ajuta la implementarea politicii de securitate prin
    a. Restricționarea coloanelor dintr-un tabel la care are acces un utilizator
    b. Restricționarea bazelor de date la care are acces un utilizator
    c. Restricționarea rândurilor dintr-un tabel la care are acces un utilizator
    d. Stocarea rezultatelor auditării bazei de date
    e. Monitorizarea intrușilor la baza de date

15. Rolurile sunt create în Microsoft SQL Server și Sybase Adaptive Server utilizând
    a. Procedura memorată sp_create_role
    b. Procedura memorată sp_add_role
    c. Procedura memorată sp_addrole
    d. Instrucțiunea CREATE ROLE
    e. Instrucțiunea GRANT
16. Rolurile sunt create în Oracle folosind
    a. Procedura memorată sp_create_role
    b. Procedura memorată sp_add_role
    c. Procedura memorată sp_addrole
    d. Instrucțiunea CREATE ROLE
    e. Instrucțiunea GRANT
17. Privilegiile de roluri sunt atribuite utilizatorilor de baze de date în Microsoft SQL Server și Sybase Adaptive Server folosind
    a. Procedura memorată sp_create_role_member
    b. Procedura memorată sp_add_role_member
    c. Procedura memorată sp_addrolemember
    d. Instrucțiunea CREATE ROLE MEMBER
    e. Instrucțiunea GRANT
18. Privilegiile de roluri sunt atribuite utilizatorilor de baze de date în Oracle folosind
    a. Procedura memorată sp_create_role_member
    b. Procedura memorată sp_add_role_member
    c. Procedura memorată sp_addrolemember
    d. Instrucțiunea CREATE ROLE MEMBER
    e. Instrucțiunea GRANT
19. Scrieți instrucțiunea SQL pentru a acorda utilizatorilor manager_1 și manager_2 privilegiile SELECT, INSERT și DELETE pentru rândurile din tabelul EMPLOYEES. Dacă vreți să testați instrucțiunea, la început va trebui să creați utilizatorii manager_1 și manager_2.
20. Scrieți instrucțiunea SQL pentru a anula privilegiile INSERT, UPDATE și DELETE ale utilizatorului clerk_127 la tabelul MOVIE. Dacă vreți să testați instrucțiunea, la început va trebui să creați utilizatorul clerk_127.

CAPITOLUL 9

Menținerea integrității bazei de date cu ajutorul tranzacțiilor

În capitolul următor, vom analiza modul în care limbajul SQL poate fi incorporat în programele de aplicații, cum sunt cele scrise în Java, Microsoft .NET Framework, C și alte limbaje /medii de programare. Însă înainte de această analiză, trebuie să înțelegeți conceptul de tranzacții cu baza de date, deoarece veți avea nevoie de ele pentru a implementa corect procesele de afaceri care actualizează baza de date.

Ce este o tranzacție cu baza de date?

O *tranzacție* (*transaction*) este o serie discretă de acțiuni care trebuie sau să fie procesate complet, sau să nu fie procesate deloc. Tranzacția mai este denumită *unitate de lucru* (*unit of work*) pentru a evidenția caracterul său „totul sau nimic". Proprietățile tranzacțiilor pot fi reținute ușor cu ajutorul acronimului ACID (Atomicitate, Consistență, Izolare, Durabilitate):

- **Atomicitate** O tranzacție trebuie să rămână un întreg. Aceasta înseamnă că trebuie să fie efectuată complet sau deloc. În cazul reușitei, toate modificările efectuate de tranzacție trebuie păstrate de sistem. În cazul eșecului unei tranzacții, toate modificările efectuate de tranzacția respectivă trebuie anulate complet. În sistemele de baze de date, folosim termenul *refacere* sau *restaurare* (*rollback*) pentru procesul care anulează toate modificările efectuate de o tranzacție nereușită și termenul *finalizare* (*commit*) pentru procesul care face permanente modificările tranzacției.
- **Consistență** O tranzacție ar trebui să transforme baza de date dintr-o stare consistentă în altă stare consistentă. De exemplu, o tranzacție care creează o factură pentru o comandă transformă comanda dintr-o comandă *înregistrată în documente* într-o comandă *facturată*, incluzând toate modificările corespunzătoare în baza de date.
- **Izolare** Fiecare tranzacție trebuie să efectueze acțiunile proprii independent de orice altă tranzacție care ar putea avea loc în același timp.
- **Durabilitate** Modificările efectuate de tranzacțiile finalizate ar trebui să rămână permanente, chiar și după o oprire ulterioară a sistemului sau o defecțiune a bazei de date ori a altei componente importante din sistem. În terminologia obiectelor, pentru datele stocate permanent este folosit termenul *persistență* sau *durabilitate*. Conceptul „permanent" poate părea confuz în această frază, deoarece într-o bază de date OLTP (online transaction processing)[1] nimic nu pare să rămână stabil. Rețineți următoarele: *permanent* înseamnă că schimbarea nu va dispărea atunci când baza de date este oprită sau se defectează – nu înseamnă că datele sunt într-o stare permanentă care nu mai poate fi schimbată.

Suportul pentru tranzacții în sistemele DBMS relaționale

În afara sistemelor personale de baze de date, majoritatea sistemelor DBMS relaționale oferă suport pentru tranzacții. Acesta include anumite resurse în SQL pentru identificarea începutului și sfârșitului fiecărei tranzacții, precum și facilitatea de a înregistra toate modificările efectuate de tranzacții astfel încât să poată fi efectuată o restaurare atunci

[1] online transaction processing = prelucrare online a tranzacțiilor

când este necesar. După cum puteți intui, standardele au rămas în urma nevoii de suport pentru tranzacții, astfel că tipul de suport variază între producătorii de sisteme RDBMS.

Majoritatea sistemelor RDBMS înregistrează toate tranzacțiile și modificările efectuate de acestea într-un *jurnal de tranzacții*. Imaginea anterioară și viitoare a fiecărei modificări în baza de date efectuată de o tranzacție este înregistrată în jurnalul de tranzacții. Acesta facilitează orice restaurare necesară, pentru că imaginile anterioare pot fi utilizate pentru a anula modificările efectuate de tranzacție în baza de date. Finalizarea unei tranzacții nu este completă până când înregistrarea finalizării nu a fost scrisă în jurnalul de tranzacții. Deoarece modificările în baza de date nu sunt scrise întotdeauna imediat pe disc, jurnalul de tranzacții reprezintă uneori singura modalitate de recuperare a datelor în cazul unei defecțiuni a sistemului.

Să analizăm în continuare suportul pentru tranzacții în câteva dintre cele mai obișnuite produse RDBMS utilizate în prezent.

Suportul pentru tranzacții în Microsoft SQL Server

Microsoft SQL Server suportă tranzacțiile în trei moduri: finalizare automată (*autocommit*), explicit și implicit. Toate cele trei moduri sunt disponibile atunci când vă conectați direct la baza de date folosind un instrument client proiectat special în acest scop. Însă dacă intenționați să folosiți un driver ODBC sau JDBC, ar trebui să consultați documentația driverului pentru a afla informații privind suportul pentru tranzacții. Cele trei moduri sunt:

- **Modul de finalizare automată** În acest mod, fiecare instrucțiune SQL este executată automat până la finalizare. În esență, acest mod determină ca fiecare instrucțiune SQL să fie o instrucțiune distinctă. Fiecare conexiune la Microsoft SQL Server folosește modul de finalizare automată până când este inițiată o tranzacție explicită sau este configurat modul de tranzacții implicit. Cu alte cuvinte, modul de finalizare automată este modul prestabilit de lucru cu tranzacții pentru orice conexiune SQL Server.
- **Modul explicit** În acest mod, fiecare tranzacție începe cu o instrucțiune BEGIN TRANSACTION și se încheie fie cu o instrucțiune COMMIT TRANSACTION (pentru încheierea cu succes), fie cu o instrucțiune ROLLBACK TRANSACTION (în caz de nereușită). Acest mod este folosit deseori în programele de aplicații, în procedurile memorate, în procedurile de declanșare și în scripturi. Sintaxa generală a celor trei instrucțiuni SQL este următoarea:

```
BEGIN TRAN[SACTION] [tran_name | @tran_name_variable]
 [WITH MARK [,description']]

COMMIT [TRAN[SACTION] [tran_name | @tran_name_variable]]

ROLLBACK [TRAN[SACTION] [tran_name | @tran_name_variable |
          savepoint_name | @savepoint_name_variable]]
```

- **Modul implicit** Modul implicit pentru tranzacții este activat sau dezactivat cu ajutorul comenzii SET IMPLICIT_TRANSACTIONS {ON|OFF}. Atunci când este activat modul implicit, este inițiată o tranzacție nouă ori de câte ori este executată o listă cu anumite instrucțiuni SQL, incluzând, printre altele, DELETE, INSERT, SELECT și UPDATE. După demararea în mod implicit a unei tranzacții, aceasta continuă până când este finalizată sau este anulată. Dacă utilizatorul bazei de date se deconectează înainte de specificarea unei instrucțiuni de încheiere a tranzacției, tranzacția este anulată în mod automat.

Suportul pentru tranzacții în Sybase Adaptive Server

Suportul pentru tranzacții din Sybase Adaptive Server este foarte asemănător cu cel din Microsoft SQL Server, însă există câteva diferențe. Sybase Adaptive Server suportă tranzacții în două moduri: finalizare automată și explicit. Ambele moduri sunt disponibile atunci când vă conectați direct la baza de date folosind un instrument client proiectat special în acest scop. Însă dacă intenționați să folosiți un driver ODBC sau JDBC, ar trebui să consultați documentația driverului pentru a afla informații privind suportul pentru tranzacții. În continuare este prezentată o descriere a celor două moduri:

- **Modul de finalizare automată** În acest mod, fiecare instrucțiune SQL este finalizată automat după terminare. În esență, acest mod determină ca fiecare instrucțiune SQL să fie o instrucțiune distinctă. Orice conexiune la Sybase Adaptive Server folosește modul de finalizare automată până când este inițiată o tranzacție explicită. Cu alte cuvinte, modul de finalizare automată este modul prestabilit pentru tranzacții pentru orice conexiune.
- **Modul explicit** În acest mod, fiecare tranzacție începe cu o instrucțiune BEGIN TRANSACTION și se încheie fie cu o instrucțiune COMMIT TRANSACTION (pentru încheierea cu succes), fie cu o instrucțiune ROLLBACK TRANSACTION (în caz de nereușită). Există și un *punct de salvare* (*savepoint*), care asigură un punct consistent în procesarea unei tranzacții și poate fi folosit într-o restaurare. Modul explicit este folosit cel mai des în programele de aplicații, în procedurile memorate, în procedurile de declanșare și în scripturi. Sintaxa generală a instrucțiunilor SQL care suportă tranzacții este următoarea:

```
BEGIN TRAN[SACTION] transaction_name

SAVE TRAN[SACTION] savepoint_name

COMMIT TRAN | TRANSACTION | WORK [transaction_name]

ROLLBACK [TRAN | TRANSACTION | WORK]
         [transaction_name | savepoint_name]
```

Suportul pentru tranzacții în Oracle

Oracle acceptă doar două moduri pentru tranzacții: finalizare automată și implicit. La fel ca în cazul sistemelor Microsoft SQL Server și Sybase Adaptive Server, suportul diferă atunci când sunt folosite drivere ODBC și JDBC, astfel că în cazurile respective este recomandată consultarea documentației furnizate împreună cu driverul. Cele două moduri pentru tranzacții din Oracle sunt următoarele:

- **Modul de finalizare automată** La fel ca în alte sisteme DBMS, în acest mod fiecare instrucțiune SQL este finalizată automat după terminarea rulării. Însă în Oracle modul de finalizare automată *nu* este modul prestabilit. pentru tranzacții pentru orice conexiune. Modul de finalizare automată este activat și dezactivat cu ajutorul comenzii SET AUTOCOMMIT (inițial este dezactivat):

```
SET AUTOCOMMIT ON

SET AUTOCOMMIT OFF
```

- **Modul implicit** O tranzacție este începută implicit atunci când utilizatorul bazei de date se conectează la baza de date (atunci când începe o sesiune nouă cu baza de date). Acesta este modul prestabilit pentru tranzacții în Oracle. Când o tranzacție este încheiată printr-o finalizare sau printr-o anulare, este inițiată automat o tranzacție nouă. Spre deosebire de Microsoft SQL Server și Sybase Adaptive Server, nu sunt permise tranzacțiile imbricate (tranzacții în interiorul altor tranzacții). O tranzacție se încheie prin finalizare în următoarele situații: 1) utilizatorul bazei de date execută instrucțiunea SQL COMMIT; 2) sesiunea la baza de date se încheie normal (utilizatorul inițiază o comandă EXIT); 3) 1) utilizatorul bazei de date inițiază o instrucțiune SQL DDL (adică o instrucțiune CREATE, DROP sau ALTER). O tranzacție se încheie cu o anulare în următoarele situații: 1) utilizatorul bazei de date inițiază instrucțiunea SQL ROLLBACK; 2) sesiunea la baza de date se încheie anormal (conexiunea clientului este anulată sau baza de date se blochează ori se oprește prin una dintre opțiunile de oprire care întrerupe conexiunile client în loc să aștepte încheierea acestora).

Suportul pentru tranzacții în MySQL

Suportul pentru tranzacții a fost adăugat în MySQL începând cu versiunea 3.23.0. Pentru a asigura compatibilitatea versiunilor mai vechi cu cele care acceptă suportul pentru tranzacții, acest suport a fost adăugat folosindu-se motoare de stocare noi. Tabelele create cu ajutorul motorului de stocare implicit (ISAM, denumit MyISAM în versiunile MySQL mai noi) nu au suport pentru tranzacții. Pentru a permite acest suport, trebuie specificat la crearea tabelului motorul de stocare InnoDB sau BDB (Berkeley DB). De exemplu:

```
CREATE TABLE LANGUAGE
 (LANGUAGE_CODE          CHAR(2)       NOT NULL,
  LANGUAGE_NAME          VARCHAR(40)   NOT NULL,
 PRIMARY KEY (LANGUAGE_CODE))
 ENGINE = INNODB;
```

Există multe implicații ale utilizării unor motoare diferite de stocare; ca urmare, este recomandat să consultați documentația versiunii MySQL pe care o utilizați înainte de a lua o decizie.

Pentru tabelele create folosind motoarele de stocare InnoDB sau BDB, suportul pentru tranzacții este disponibil în două moduri:

- **Modul de finalizare automată** Modul de finalizare automată este activat și dezactivat cu ajutorul unei instrucțiuni SET în care valoarea 0 dezactivează modul de lucru, iar valoarea 1 îl activează:

```
SET AUTOCOMMIT=0;

SET AUTOCOMMIT=1;
```

- **Modul implicit** Acest mod devine activ ori de câte ori modul de finalizare automată este dezactivat. În modul implicit pentru tranzacții pot fi folosite următoarele instrucțiuni SQL:

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT];

SAVEPOINT savepoint_identifier;

ROLLBACK [TO savepoint_identifier];

COMMIT;
```

Suportul pentru tranzacții în DB2 UDB

Toate tranzacțiile din sistemul DB2 UDB (Universal Database) sunt implicite. Comportamentul este destul de asemănător suportului pentru tranzacții din Oracle. Instrucțiunile SQL utilizate pentru acest suport sunt următoarele:

```
COMMIT [WORK];

SAVEPOINT savepoint_name [options];

RELEASE [TO] SAVEPOINT savepoint_name;

ROLLBACK [WORK] [TO SAVEPOINT [savepoint_name]];
```

Blocarea și interblocarea tranzacțiilor

Deși partajarea simultană a datelor între mai mulți utilizatori ai unei baze de date are avantaje semnificative, există și un dezavantaj important care poate provoca pierderea actualizărilor. Din fericire, producătorii de baze de date au găsit soluții pentru această problemă. În secțiunea curentă sunt prezentate problema actualizării concurente și diferite soluții.

Problema actualizării concurente

Figura 9-1 ilustrează problema actualizării concurente, care se manifestă atunci când mai multe sesiuni ale bazei de date au permisiunea de a actualiza concurent aceleași date. Amintiți-vă că o sesiune este creată de fiecare dată când un utilizator se conectează la baza de date, inclusiv când același utilizator se conectează de mai multe ori la baza de date. Problema actualizării concurente apare cel mai des între doi utilizatori diferiți ai bazei de date, care nu știu că efectuează actualizări conflictuale ale acelorași date. Și un utilizator al bazei de date cu mai multe conexiuni poate greși dacă aplică actualizări folosind mai multe sesiuni la baza de date.

În scenariul prezentat este folosită o companie fictivă care vinde produse și creează o factură pentru fiecare comandă livrată. Figura 9-1 ilustrează Utilizatorul A, un angajat din departamentul de livrări care pregătește o factură pentru un client; aceasta presupune actualizarea datelor privitoare la client prin adăugarea sumei la soldul debitor. În același timp, Utilizatorul B, vânzător în departamentul de încasări, prelucrează o plată de la același client, operație care presupune actualizarea soldului debitor al clientului prin scăderea sumei plătite. În continuare este prezentată secvența de operații, așa cum este ilustrată în figura 9-1:

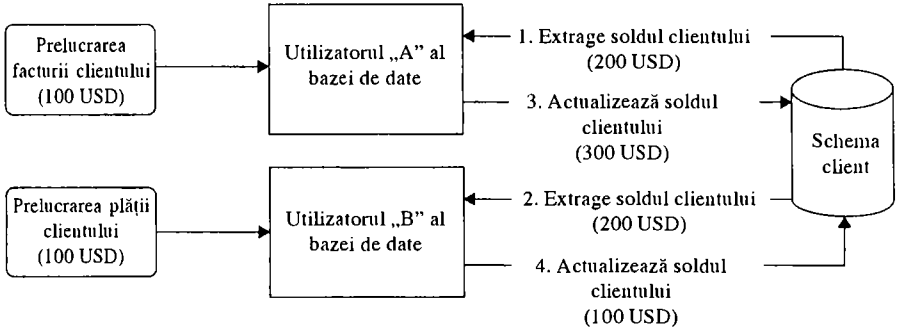


Figura 9-1 Problema actualizării concurente

1. Utilizatorul A interoghează baza de date și extrage soldul debitor al clientului, care are valoarea de 200 USD.
2. Câteva secunde mai târziu, Utilizatorul B interoghează baza de date și extrage același sold debitor al clientului, care are tot valoarea de 200 USD.
3. După alte câteva secunde, Utilizatorul A aplică actualizarea proprie, adăugând factura de 100 USD la soldul debitor al clientului, ceea ce conduce la un sold nou în baza de date (300 USD).
4. În final, Utilizatorul B aplică actualizarea sa, scăzând plata de 100 USD din soldul debitor extras din baza de date (200 USD), rezultând astfel un sold nou în baza de date (100 USD). Acest utilizator nu știe nimic de actualizarea efectuată de Utilizatorul A și ca urmare setează (incorect) valoarea soldului debitor la 100 USD.

Soldul debitor al acestui client ar trebui să fie 200 USD, însă actualizarea efectuată de Utilizatorul A a fost suprascrisă de actualizarea efectuată de Utilizatorul B. Compania pierde urma celor 100 USD, care fie rămân pierduți, fie vor necesita mult timp pentru a fi descoperiți și corectați. După cum puteți vedea, permiterea actualizărilor concurente în baza de date fără nici un fel de control poate provoca pierderea unor actualizări. Majoritatea producătorilor de baze de date implementează o strategie de blocare pentru a preveni actualizarea concurentă a acelorași date.

Mecanismele de blocare

O *blocare* (*lock*) este un element de control plasat în baza de date pentru reținerea datelor astfel încât un singur utilizator să le poată actualiza. Atunci când datele sunt blocate, nici o altă sesiune a bazei de date nu le poate actualiza până când nu este anulată blocarea, eveniment care se petrece de obicei printr-o instrucțiune SQL COMMIT sau ROLLBACK. Orice altă sesiune care încearcă să actualizeze datele blocate va fi plasată într-o stare de așteptare (*lock wait*), iar sesiunea va fi întreruptă până la anularea blocării. Unele sisteme de baze de date, cum este DB2, produs de IBM, vor întrerupe o sesiune care așteaptă prea mult timp și vor returna un mesaj de eroare în loc să finalizeze actualizarea solicitată. Alte sisteme, cum este Oracle, pot lăsa o sesiune în starea de așteptare pentru o perioadă nedefinită de timp.

Nu ar trebui să vă surprindă faptul că există diferențe semnificative între modurile de tratare a blocărilor implementate în diferite sisteme de baze de date. În această secțiune este prezentată o analiză generală, împreună cu recomandarea de a consulta documentația producătorului bazei de date pentru a afla detalii referitoare la suportul pentru blocări. Blocările pot fi produse la diferite niveluri (funcție denumită *granularitate de blocare*), iar unele sisteme de baze de date, între care se numără Sybase Adaptive Server, Microsoft SQL Server și DB2, acceptă mai multe niveluri prin *extinderea automată a blocării* (*lock escalation*), care ridică blocările la niveluri superioare pe măsură ce o sesiune a bazei de

date plasează din ce în ce mai multe blocări asupra acelorași obiecte din baza de date. Blocarea și deblocarea unor cantități mici de date necesită o supraîncărcare semnificativă, astfel că extinderea blocărilor la niveluri superioare poate îmbunătăți substanțial performanța. Nivelurile obișnuite de blocare sunt următoarele:

- **Baza de date** Întreaga bază de date este blocată, astfel încât numai o singură sesiune a bazei de date poate să aplice actualizări. Evident, este o situație extremă care nu ar trebui să se producă foarte des, însă poate fi utilă atunci când sunt efectuate activități importante de mentenanță, cum ar fi actualizarea la o versiune nouă a software-ului pentru baza de date. Oracle suportă acest nivel indirect, atunci când baza de date este deschisă în mod exclusiv, ceea ce restricționează baza de date la o singură sesiune utilizator.
- **Fișier** Este blocat un întreg fișier din baza de date. Amintiți-vă că un fișier poate conține părți dintr-un tabel, un tabel întreg sau părți din mai multe tabele. Acest nivel nu este preferat în bazele de date moderne, deoarece datele blocate pot fi foarte diferite.
- **Tabel** Este blocat un tabel întreg. Acest nivel este util dacă efectuați o modificare la nivelul unui tabel, cum ar fi reîncărcarea tuturor datelor din tabel, actualizarea tuturor rândurilor sau modificarea structurii unui tabel prin adăugarea sau ștergerea de coloane. Oracle denumește acest nivel *blocare DDL* (*DDL lock*), iar nivelul este folosit atunci când sunt inițiate instrucțiuni DDL (CREATE, DROP și ALTER) corespunzătoare unui tabel sau unui alt obiect din baza de date.
- **Bloc sau pagină** Este blocat un bloc sau este blocată o pagină dintr-un fișier al bazei de date. *Blocul* este cea mai mică unitate de date pe care sistemul de operare o poate citi sau scrie într-un fișier. În majoritatea calculatoarelor personale, dimensiunea blocului este numită *dimensiune de sector*. Unele sisteme de operare folosesc pagini în locul blocurilor. O *pagină* este un bloc virtual de dimensiune fixă, de obicei 2K sau 4K, utilizat pentru simplificarea procesării în cazul în care există mai multe dispozitive de stocare ce suportă dimensiuni diferite de blocuri. Sistemul de operare poate să citească și să scrie paginile și permite driverelor hardware să transfere paginile în blocurile corespunzătoare. La fel ca în cazul blocării fișierelor, nici blocarea la nivel de bloc (pagină) nu este preferată în sistemele moderne de baze de date, datorită diversității datelor care pot fi scrise în același bloc din fișier.
- **Rând** Este blocat un rând dintr-un tabel. Este cel mai folosit nivel de blocare; practic toate sistemele moderne de baze de date suportă acest nivel.
- **Coloană** Sunt blocate unele coloane dintr-un rând al unui tabel. Această metodă pare extraordinară în teorie, însă nu este foarte practică datorită resurselor necesare pentru activarea și anularea blocărilor la acest nivel de granularitate. În sistemele moderne de baze de date există un suport slab pentru acest nivel.

Blocările sunt activate întotdeauna atunci când sunt actualizate sau șterse date. De asemenea, majoritatea sistemelor RDBMS suportă utilizarea clauzei FOR UPDATE OF într-o instrucțiune SELECT pentru a permite activarea blocărilor atunci când utilizatorul bazei de date declară *intenția* de a actualiza ceva. Unele blocări pot fi considerate *exclusiv pentru citire*, ceea ce împiedică alte sesiuni să citească datele blocate. Numeroase sisteme RDBMS au parametrii de sesiune care pot fi setați astfel încât să ajute la controlul comportamentului blocărilor. Unul dintre comportamentele de blocare care trebuie luat în considerare este dacă toate rândurile extrase cu ajutorul unui cursor sunt blocate până la următoarea comandă COMMIT sau ROLLBACK, ori dacă rândurile citite anterior sunt eliberate la extragerea următorului rând. Pentru a afla mai multe detalii, consultați documentația producătorului bazei de date.

Principala problemă a mecanismelor de blocare este faptul că blocările produc *conflicte*, ceea ce înseamnă că activarea unei blocări în scopul prevenirii pierderii de date în urma actualizărilor concurente are ca efect secundar declanșarea unor conflicte între sesiunile concurente pentru obținerea dreptului de a aplica actualizări. În cel mai bun caz, conflictele de blocare încetinesc procesele utilizatorilor datorită faptului că sesiunile așteaptă anularea blocărilor. În cel mai rău caz, cererile concurente de blocare pot întrerupe sesiunile pentru perioade nedefinite de timp, după cum veți afla în sesiunea următoare.

Interblocările

O interblocare (*deadlock*) este o situație în care două sau mai multe sesiuni la baza de date au blocat anumite date, iar apoi fiecare dintre ele solicită o blocare a datelor blocate de cealaltă sesiune. Figura 9-2 ilustrează această situație.

În acest exemplu sunt folosiți doi utilizatori dintr-o companie fictivă, denumiți A și B. Utilizatorul A este un angajat din departamentul de servicii pentru clienți și încearcă să corecteze o plată care a fost creditată în contul altui client. Acest utilizator trebuie să scadă

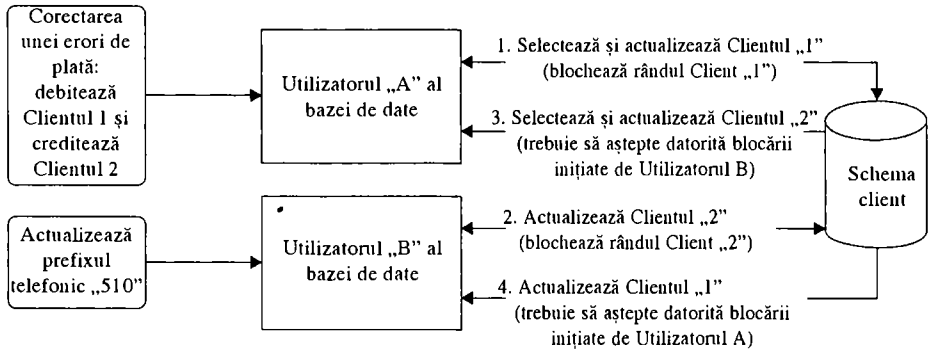


Figura 9-2 Interblocarea

suma plătită de la Clientul 1 și să o adauge la Clientul 2. Utilizatorul B este un specialist în baze de date din departamentul IT și a scris o instrucțiune SQL pentru a actualiza numerele de telefon ale unor clienți ale căror prefixe au fost modificate recent de compania telefonică. Instrucțiunea conține o clauză WHERE care limitează actualizarea doar la clienții care au numere de telefon cu prefixul 510 și actualizează acele numere cu noul prefix. Utilizatorul B inițiază instrucțiunea SQL în timp ce Utilizatorul A rezolvă problema cu plata eronată. Clienții 1 și 2 au ambii numere de telefon care trebuie actualizate. Secvența de evenimente (totul se întâmplă pe parcursul a câteva secunde), așa cum este ilustrată în figura 9-2, este următoarea:

1. Utilizatorul A selectează datele corespunzătoare Clientului 1 și aplică o actualizare pentru a scădea soldul debitor. Nu este inițiată încă nici o finalizare pentru că aceasta este doar o parte din tranzacția care trebuie efectuată. Rândul corespunzător Clientului 1 este blocat acum datorită actualizării.
2. Instrucțiunea inițiată de Utilizatorul B actualizează numărul de telefon corespunzător Clientului 2. Întreaga instrucțiune SQL trebuie să ruleze ca o singură tranzacție, astfel încât nu este efectuată nici o finalizare în acest moment; ca urmare, Utilizatorul B menține blocarea rândului corespunzător Clientului 2.
3. Utilizatorul A selectează soldul debitor al Clientului 2 și apoi inițiază o actualizare pentru a mări soldul debitor (cu aceeași sumă scăzută din contul Clientului 1). Solicitarea trebuie să aștepte pentru că Utilizatorul B menține blocarea rândului care trebuie actualizat.
4. Instrucțiunea inițiată de Utilizatorul B încearcă acum să actualizeze numărul de telefon al Clientului 1. Actualizarea trebuie să aștepte pentru că Utilizatorul A menține blocarea rândului care trebuie actualizat.

Aceste două sesiuni la baza de date sunt acum în interblocare. Utilizatorul A nu poate continua datorită blocării menținute de Utilizatorul B, și viceversa. Teoretic, aceste două sesiuni la baza de date vor rămâne blocate la infinit. Din fericire, sistemele DBMS moderne includ funcții pentru tratarea acestei situații. O metodă este prevenirea interblocărilor. Doar câteva sisteme DBMS au această funcție, datorită resurselor suplimentare considerabile necesare și a imposibilității practice de prezicere a următoarei operații pe care o va efectua un utilizator interactiv al bazei de date. Teoretic, trebuie inspectate toate cererile de blocare pentru a analiza care dintre ele poate să provoace conflicte și pentru a nu permite activarea blocării respective dacă este posibilă apariția unei interblocări. O abordare mai obișnuită a problemei este detectarea interblocărilor, care anulează una dintre cererile care au provocat interblocarea. Aceasta poate fi realizată fie prin cronometrarea duratei de așteptare a unei blocări și de anulare a acesteia după un anumit interval de timp prestabilit, fie prin inspectarea periodică a tuturor blocărilor pentru a detecta două sesiuni care se blochează reciproc. În ambele cazuri, una dintre cereri trebuie să fie anulată, iar modificările tranzacției începute să fie anulate pentru a permite satisfacerea celeilalte cereri.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. O tranzacție
   a. Poate fi procesată parțial
   b. Nu poate fi procesată parțial
   c. Schimbă baza de date dintr-o stare consistentă în alta
   d. Este denumită uneori *unitate de lucru*
   e. Are proprietățile descrise de acronimul ACID
2. Litera *A* din acronimul ACID provine de la
   a. Automat
   b. Auxiliar
   c. Atomicitate
   d. Augmentat
   e. Administrare
3. Litera *C* din acronimul ACID provine de la
   a. Corelat
   b. Consistență
   c. Codificare
   d. Calculat
   e. Consolidat
4. Litera *I* din acronimul ACID provine de la
   a. Integrare
   b. Instrucțiune
   c. Iconic
   d. Izolare
   e. Informație

5. Litera *D* din acronimul ACID provine de la
   a. Durabilitate
   b. Dedicație
   c. Dualitate
   d. Date
   e. Distribuție
6. Procesul care anulează modificările efectuate de o tranzacție eșuată este numit
   a. Înregistrarea tranzacției
   b. Finalizare
   c. Refacere
   d. Recuperare
   e. Crearea unui punct de salvare
7. Procesul care face permanente modificările efectuate de o tranzacție este numit
   a. Înregistrarea tranzacției
   b. Finalizare
   c. Refacere
   d. Crearea unui punct de salvare
   e. Salvarea tranzacției
8. Suportul pentru tranzacții în bazele de date relaționale include:
   a. Identificarea începutului fiecărei tranzacții
   b. Identificarea sfârșitului fiecărei tranzacții
   c. Managementul bazei de date distribuite
   d. Salvările periodice ale bazei de date
   e. Jurnalul de tranzacții
9. Microsoft SQL Server suportă următoarele moduri de tranzacții:
   a. Finalizare automată
   b. Automat
   c. Durabil

d. Explicit

e. Implicit

10. Oracle suportă următoarele moduri de tranzacții:

a. Finalizare automată

b. Automat

c. Durabil

d. Explicit

e. Implicit

11. În modul implicit pentru tranzacții din Microsoft SQL Server, o tranzacție nouă este inițiată de

a. Conectarea la baza de date

b. O instrucțiune COMMIT

c. O instrucțiune ROLLBACK

d. O instrucțiune INSERT

e. O instrucțiune SELECT

12. În modul implicit pentru tranzacții din Oracle, o tranzacție nouă este inițiată de

a. Conectarea la baza de date

b. O instrucțiune COMMIT

c. O instrucțiune ROLLBACK

d. O instrucțiune INSERT

e. O instrucțiune SELECT

13. Instrucțiunile SQL utilizate pentru gestionarea tranzacțiilor în Microsoft SQL Server și Sybase Adaptive Server sunt

a. BEGIN TRANSACTION

b. END TRANSACTION

c. COMMIT

d. ROLLBACK

e. SET AUTOCOMMIT

14. Instrucțiunile SQL utilizate pentru gestionarea tranzacțiilor în Oracle sunt
    a. BEGIN TRANSACTION
    b. END TRANSACTION
    c. COMMIT
    d. ROLLBACK
    e. SET AUTOCOMMIT
15. În MySQL, suportul pentru tranzacții
    a. Se aplică doar motoarelor de stocare ISAM și MyISAM
    b. Se aplică doar motoarelor de stocare InnoDB și BDB
    c. Include modurile finalizare automată, implicit și explicit
    d. Include modurile finalizare automată și implicit
    e. Include modurile finalizare automată și explicit
16. Instrucțiunile SQL utilizate pentru gestionarea tranzacțiilor în DB2 UDB sunt
    a. BEGIN TRANSACTION
    b. END TRANSACTION
    c. COMMIT
    d. ROLLBACK
    e. SAVEPOINT
17. Cantitatea de date afectată de o blocare (granularitatea blocării) poate fi
    a. O bază de date
    b. Un tabel
    c. Un rând
    d. O coloană
    e. Un bloc sau o pagină
18. Problema actualizării concurente
    a. Este o consecință a partajării simultane a datelor
    b. Nu poate apărea când este activată (ON) opțiunea AUTOCOMMIT
    c. Este motivul pentru care trebuie asigurat suport pentru blocarea tranzacțiilor

d. Apare când doi utilizatori ai bazei de date inițiază instrucțiuni SELECT care intră în conflict

e. Apare când doi utilizatori ai bazei de date efectuează actualizări conflictuale aceloraşi date

19. O blocare

a. Este un element de control atașat datelor pentru a le rezerva în scopul actualizării de către utilizator

b. Este anulată de obicei atunci când are loc o operație COMMIT sau ROLLBACK

c. Are setat un interval de așteptare în sistemul DB2 și în alte produse RDBMS

d. Poate provoca apariția unor conflicte atunci când alți utilizatori încearcă să actualizeze datele blocate

e. Poate avea niveluri și un protocol de extindere în unele produse RDBMS

20. O interblocare

a. Este o blocare ce a depășit intervalul de așteptare și din acest motiv nu mai este necesară

b. Apare când doi utilizatori ai bazei de date solicită simultan blocarea unor date blocate de celălalt utilizator

c. Poate, teoretic, să plaseze doi sau mai mulți utilizatori într-o stare de așteptare infinită

d. Poate fi rezolvată prin detectarea interblocărilor în unele sisteme RDBMS

e. Poate fi rezolvată prin intervale de așteptare a blocării în unele sisteme RDBMS

CAPITOLUL 10

Integrarea limbajului SQL în aplicații

În acest capitol, vom analiza modul în care limbajul SQL este folosit în aplicații. O *aplicație* este un set de programe de calculator proiectate pentru a rezolva o anumită problemă, cum ar fi un sistem de introducere a comenzilor, un sistem de procesare a statelor de plată sau un sistem de contabilitate. Aplicațiile pot fi construite folosind un limbaj de programare procedural de uz general, cum ar fi Java, C sau C++, ori un mediu de programare de genul Microsoft .NET Framework. Un limbaj de programare *procedural* specifică o secvență explicită de etape care trebuie parcurse pentru a finaliza o tranzacție. În aplicațiile care folosesc baze de date relaționale, dezvoltatorii de aplicații utilizează diferite tehnici pentru comunicarea cu baza de date, care, desigur, includ transmiterea de instrucțiuni SQL către baza de date pentru procesarea și manipularea rezultatelor. Acest capitol prezintă o parte dintre aceste tehnici.

Procesarea cursorului

Limbajele de programare procedurale sunt proiectate să trateze o singură înregistrare la un moment dat (o instanță de obiect sau o colecție de obiecte în cazul limbajelor orientate pe obiecte, cum este Java). Această regulă creează situații dificile atunci când limbajul SQL este folosit împreună cu un limbaj de programare, pentru că interogările SQL produc de regulă seturi de rezultate care conțin mai multe înregistrări (rânduri) de date. Este o dereglare care trebuie rezolvată.

Pentru a depăși problema, majoritatea bazelor de date relaționale suportă conceptul de *cursor*, care este în principal un indicator către un singur rând din setul de rezultate. În Oracle, suportul pentru cursor este inclus într-o extensie SQL a limbajului procedural denumită PL/SQL (Procedural Language/SQL), iar în mod similar este inclus în Transact-SQL în sistemele Sybase Adaptive Server și Microsoft SQL Server. Aceste extensii SQL procedurale sunt prezentate în alte secțiuni din capitol. Utilizarea cursorului poate fi comparată cu folosirea unui fișier tradițional simplu (nestructurat) prin faptul că acest cursor trebuie definit și deschis înainte să poată fi utilizat, poate fi citit prin extragerea rândurilor într-o buclă de program și trebuie să fie închis atunci când programul nu mai are nevoie de el.

Instrucțiunea DECLARE CURSOR

Un cursor trebuie să fie declarat înainte de a putea fi referit de alte instrucțiuni SQL. În continuare este prezentată sintaxa generală a instrucțiunii DECLARE CURSOR, care definește un cursor:

```
DECLARE nume_cursor CURSOR FOR
instrucțiune_select
[FOR UPDATE [OF nume_coloană [, nume_coloană ...]]]
```

NOTĂ:

- *Sintaxa exactă va fi diferită în funcție de sistemul DBMS. De exemplu, standardul SQL specifică cuvântul cheie FOR, însă sintaxa Oracle folosește cuvântul cheie IS, iar sintaxa Microsoft SQL Server consideră cuvântul cheie opțional. De asemenea, Oracle solicită apariția cuvântului cheie CURSOR înainte de numele cursorului și nu după acesta. Ca în toate situațiile, pentru detalii consultați documentația sistemului DBMS propriu.*
- *Instrucțiunea DECLARE CURSOR nu are decât rolul de a defini cursorul – nu sunt selectate date până când cursorul nu este deschis și folosit. Este o procedură foarte asemănătoare cu definirea unui fișier într-un limbaj de programare tradițional – nu se întâmplă nimic până când fișierul nu este deschis și folosit.*

- *Parametrul* nume_cursor *trebuie să fie unic în cadrul programului și va fi utilizat pentru a referi cursorul declarat în instrucțiunile următoare.*
- *Instrucțiunea SQL inclusă în declarația cursorului poate conține majoritatea clauzelor instrucțiunii SELECT. Totuși, nu este recomandată folosirea clauzei INTO în acest caz, pentru că este folosită în instrucțiunea FETCH, după cum veți afla într-un alt paragraf din această secțiune.*
- *Clauza FOR UPDATE este opțională, însă în unele implementări SQL este obligatorie dacă instrucțiunile SQL ulterioare actualizează și/sau șterg datele selectate de cursor. Pentru a afla elementele specifice, consultați documentația sistemului RDBMS propriu.*

În continuare este prezentat un exemplu de instrucțiune DECLARE CURSOR scrisă în sintaxa Oracle. Instrucțiunea selectează împrumuturile de filme care au depășit termenul de înapoiere.

```
DECLARE CURSOR overdue_rentals IS
SELECT MOVIE_ID, COPY_NUMBER, TRANSACTION_ID, DUE_DATE
  FROM MOVIE_RENTAL
 WHERE DUE_DATE < CURRENT_DATE
   FOR UPDATE OF DUE_DATE;
```

Vă puteți da seama că acesta este un exemplu Oracle datorită sintaxei și utilizării parametrului CURRENT_DATE, care este o funcție Oracle ce returnează data și ora curente. În alte implementări SQL, înlocuiți expresia care determină data și ora curentă în sistemul RDBMS și corectați elementele specifice de sintaxă (dacă este necesar). De exemplu, pentru Microsoft SQL Server instrucțiunea va arăta astfel:

```
DECLARE overdue_rentals CURSOR FOR
SELECT MOVIE_ID, COPY_NUMBER, TRANSACTION_ID, DUE_DATE
  FROM MOVIE_RENTAL
 WHERE DUE_DATE < getdate()
   FOR UPDATE OF DUE_DATE;
```

Instrucțiunea OPEN CURSOR

Cursorul trebuie să fie deschis înainte de a putea fi utilizat. În acest exemplu, este posibil ca sistemul RDBMS să nu extragă nici un rând atunci când deschideți cursorul, însă pentru creșterea eficienței poate să extragă un anumit număr de rânduri și să le plaseze într-un buffer. Un *buffer* este o zonă din memoria calculatorului folosită pentru stocarea temporară a datelor. Este mult mai eficient să utilizați un buffer pentru a păstra un anumit număr de rânduri încărcate anticipat decât să accesați fișierele bazei de date pentru fiecare rând individual, deoarece calculatoarele pot accesa memoria mult mai repede decât fișierele din sistemul de fișiere. În unele cazuri însă, sistemul RDBMS *trebuie* să încarce toate

rândurile corespunzătoare unei interogări și să le sorteze înainte ca primul rând să fie returnat către programul de aplicație. Probabil ați ghicit că acestea sunt interogări care conțin o clauză ORDER BY pentru ordonarea rândurilor returnate. Dacă nu există nici un index pentru coloana indexată (coloanele indexate) folosită pentru ordonare, sistemul RDBMS trebuie să găsească și să sorteze toate rândurile înainte de a afla rândul corect și de a-l returna ca *primul rând* (cel care este sortat primul în secvența solicitată). Deși se întâmplă multe evenimente la deschiderea unui cursor, instrucțiunea propriu-zisă este simplă. Iată instrucțiunea OPEN CURSOR pentru exemplul nostru:

```
OPEN overdue_rentals;
```

Instrucțiunea FETCH

De fiecare dată când programul dumneavoastră solicită un rând nou din setul de rezultate, inițiați pur și simplu o comandă FETCH corespunzătoare cursorului. Este foarte asemănător cu citirea următoarei înregistrări dintr-un fișier într-un sistem de fișiere mai vechi, nestructurat. Amintiți-vă că un cursor este doar un indicator în setul de rezultate. De fiecare dată când este inițiată o operație de extragere, cursorul este avansat cu un rând, iar rândul indicat curent este returnat către programul apelant (adică programul care a generat comanda FETCH). Dacă nu mai există nici un rând în setul de rezultate (adică în cazul în care cursorul a fost avansat dincolo de ultimul rând din setul de rezultate), este returnat un cod către programul apelant pentru a indica acest aspect. Un alt detaliu manipulat de funcția de extragere este maparea coloanelor returnate la variabilele limbajului de programare (denumite *variabilele limbajului gazdă* sau doar *variabile gazdă*). Această operație este realizată cu ajutorul clauzei INTO și este normal ca sintaxa numelor de variabile să difere între limbajele de programare. Forma generală a instrucțiunii FETCH este următoarea:

```
FETCH nume_cursor
  [INTO listă_variabile];
```

Observați că instrucțiunea FETCH face referire doar la numele cursorului și la variabilele gazdă. Declarația cursorului asociază cursorul tabelului (tabelelor) și coloanei (coloanelor) referite. În majoritatea implementărilor SQL, variabilele gazdă trebuie să fie declarate înainte să poată fi referite într-o instrucțiune FETCH. Sintaxa și regulile pentru declararea variabilelor gazdă variază în implementări SQL diferite, astfel că este recomandat să consultați documentația producătorului pentru a afla detalii specifice. În continuare este prezentat un exemplu care va rula în majoritatea implementărilor SQL (clauza INTO a fost omisă intenționat). Sistemul Oracle solicită introducerea clauzei INTO împreună cu alte instrucțiuni care încep și încheie un bloc de instrucțiuni PL/SQL, astfel încât acest exemplu nu va funcționa în Oracle fără o serie de modificări:

```
FETCH overdue_rentals;
```

Instrucțiunile UPDATE și DELETE

Limbajul SQL oferă o metodă convenabilă de actualizare a datelor selectate cu un cursor sau de ștergere a rândurilor din tabelele referite de un cursor. Rețineți că nici o instrucțiune SQL nu poate să actualizeze sau să șteargă date în mai mult de un tabel, astfel că un cursor care conține o uniune are o serie de restricții de actualizare și de ștergere. După cum ați remarcat anterior, multe implementări SQL solicită specificarea clauzei FOR UPDATE pentru cursoarele care vor fi folosite pentru actualizări și/sau ștergeri. O formă specială a instrucțiunilor UPDATE și DELETE este folosită atunci când se lucrează cu cursoare. Această formă conține clauza WHERE CURRENT OF, care denumește cursorul. Rândul care a fost actualizat sau șters este ultimul extras de cursorul specificat. Dacă nici un rând nu a fost extras de cursor, orice instrucțiune care conține clauza WHERE CURRENT Office va eșua.

În continuare este prezentată sintaxa generală a instrucțiunii UPDATE pentru cursor, urmată de exemple Oracle și Microsoft SQL Server:

```
UPDATE nume_tabel
 SET listă_de_coloane_și_valori
 WHERE CURRENT OF nume_cursor;

UPDATE MOVIE_RENTAL
 SET DUE_DATE = CURRENT_DATE
 WHERE CURRENT OF overdue_rentals;

UPDATE MOVIE_RENTAL
 SET DUE_DATE = getdate()
 WHERE CURRENT OF overdue_rentals;
```

În continuare este prezentată sintaxa generală a instrucțiunii DELETE pentru cursor, urmată de un exemplu:

```
DELETE FROM nume_tabel
 WHERE CURRENT OF nume_cursor

DELETE FROM MOVIE_RENTAL
 WHERE CURRENT OF overdue_rentals;
```

Instrucțiunea CLOSE

Așa cum a fost menționat anterior, ar trebui să închideți cursorul întotdeauna când programul nu mai are nevoie de acesta, pentru că astfel sunt disponibilizate resursele folosite de cursor, inclusiv memoria pentru buffere. Instrucțiunea CLOSE este la fel de simplă ca instrucțiunea OPEN:

```
CLOSE overdue_rentals;
```

În Transact-SQL (Microsoft SQL Server și Sybase), memoria folosită de un cursor nu este disponibilizată complet la închiderea cursorului, astfel că este recomandat ca o practică bună de programare să anulați alocarea cursorului, așa cum este ilustrat în exemplul următor:

```
DEALLOCATE overdue_rentals;
```

Înglobarea limbajului SQL în programele de aplicații

După ce ați aflat cum tratează aplicațiile seturile de rezultate produse de interogările la bazele de date realizate cu ajutorul cursoarelor, trebuie să înțelegeți cum se conectează aplicația la baza de date și cum interacționează cu aceasta. Majoritatea conexiunilor dintre o aplicație și o bază de date folosesc o interfață API standard. O *interfață de programare a aplicațiilor* (*application programming interface* sau *API*) este un set de convenții de apel prin care un program de aplicație care accesează serviciile. Astfel de servicii pot fi oferite de sistemul de operare sau de alte produse software, cum ar fi un sistem DBMS. Interfața API pune la dispoziție un nivel de abstractizare care permite aplicației să fie portabilă între diferite sisteme de operare și diferiți producători.

Conexiunile ODBC

ODBC (Open Database Connectivity) este o interfață API standard pentru conectarea programelor de aplicație la sistemele DBMS. ODBC se bazează pe o interfață de nivel apel (Call Level Interface sau CLI, o convenție care definește modurile în care sunt făcute apelurile la servicii), care a fost definită pentru prima dată de SQL Access Group și a fost lansată pe piață în luna septembrie 1992. Deși Microsoft a fost prima companie care a realizat un produs comercial bazat pe interfața ODBC, acesta nu este un standard Microsoft, iar în prezent există versiuni disponibile pentru Unix, Macintosh și alte platforme.

ODBC este independent de orice limbaj de programare, sistem de operare sau sistem de baze de date. O aplicație scrisă pentru interfața API ODBC poate fi portată în altă bază de date sau sistem de operare prin schimbarea driverului ODBC. Driverul ODBC este cel care conectează interfața API cu baza de date sau platforma respectivă, iar o definiție cunoscută cu numele *sursă de date ODBC* conține informațiile necesare pentru ca o anumită aplicație să se conecteze la un serviciu de baze de date. În sistemele Windows, cele mai folosite drivere ODBC sunt livrate împreună cu sistemul de operare, ca și un program utilitar de definire a surselor de date ODBC (se găsesc în Control Panel sau în dosarul Administrative Tools, în funcție de versiunea Windows).

Majoritatea produselor software comerciale și cele mai multe baze de date comerciale suportă standardul ODBC, care simplifică mult mai mult sarcina producătorilor de software de a lansa pe piață și de a asigura suport pentru o varietate largă de sisteme de baze de

date. O excepție notabilă o reprezintă aplicațiile scrise în limbajul Java. Acestea folosesc o interfață API diferită, cunoscută ca JDBC, care va fi prezentată în secțiunea următoare.

O problemă comună este aceea că producătorii de baze de date relaționale nu tratează în același fel funcțiile avansate. Această problemă poate fi evitată cu ajutorul unei clauze care specifică driverului ODBC să transfere instrucțiunile SQL specifice neatinse prin interfața API ODBC. Dezavantajul acestei soluții îl reprezintă, desigur, faptul că aplicațiile ce respectă această regulă nu sunt portabile în baza de date a altui producător (iar uneori nu sunt portabile nici într-o versiune diferită a bazei de date provenite de la același producător).

Conectarea bazelor de date la aplicațiile Java

Java a fost inițial un limbaj de programare proprietar (denumit Oak) dezvoltat de compania Sun Microsystems. A devenit rapid limbajul standard de facto pentru programarea în web, cel puțin în mediile non-Microsoft. Java este un limbaj de programare orientat pe obiecte, cu verificarea strictă a tipurilor și poate fi folosit pentru a construi componente client (miniaplicații sau applets), dar și componente server (servlets). Acest limbaj are o arhitectură independentă de mașină, ceea ce determină o portabilitate foarte ridicată între platforme hardware și sisteme de operare.

Mai puteți întâlni termenii *JavaScript* și *JScript*. Acestea sunt limbaje pentru scripturi, au o sintaxă asemănătoare limbajului Java și sunt dedicate executării de funcții simple în sistemele client, cum ar fi editarea datelor. Nu sunt implementări complete ale limbajului Java și nu sunt proiectate să trateze interacțiunile cu baze de date.

JDBC (Java Database Connectivity)

JDBC (Java Database Connectivity) este o interfață API, modelată după ODBC, care are rolul de a conecta aplicațiile Java cu o varietate de produse DBMS relaționale. Unele drivere JDBC transformă interfața API JDBC în apelurile ODBC corespunzătoare și astfel se conectează la baza de date printr-o sursă de date ODBC. Alte drivere fac transferul direct în interfața API client specifică a bazei de date relaționale, cum este Oracle Call Interface (OCI). La fel ca în cazul ODBC, este disponibilă o clauză de evitare pentru trecerea instrucțiunilor SQL proprietare prin interfață. Interfața API JDBC pune la dispoziție următoarele caracteristici:

- **SQL înglobat pentru Java** Programatorii Java scriu instrucțiunile SQL ca variabile șir, șirurile sunt transferate către metodele Java, iar un procesor SQL înglobat transformă codul Java SQL în apeluri JDBC.
- **Maparea directă a tabelelor RDBMS la clasele Java** Rezultatele apelurilor SQL sunt mapate automat la variabile din clase Java. Programatorii Java pot apoi să opereze datele returnate ca obiecte Java native.

Exemplele Java sunt puțin cam complicate pentru această carte, însă puteți găsi o mulțime de astfel de exemple în site-urile web ale producătorilor care asigură suport JDBC. Astfel, site-ul Oracle Technology Network conține numeroase exemple JDBC (http://www.oracle.com/technology).

JSQL (Java SQL, SQLJ)

JSQL (denumit și Java SQL sau SQLJ) este o metodă de înglobare a instrucțiunilor SQL în Java fără a scrie nici un fel de cod special pentru a transforma instrucțiunile SQL în șiruri Java. Este o extensie a standardului ISO/ANSI pentru limbajul SQL înglobat în alte limbaje gazdă, cum este C. Un program special denumit *precompilator* este rulat pentru programul sursă și transformă automat instrucțiunile SQL scrise de programatorul Java în cod Java pur. Această metodă poate economisi o parte considerabilă din efortul necesar pentru dezvoltare.

Soluții middleware

O soluție *middleware* poate fi privită ca un sistem software care asigură medierea diferențelor dintre un program de aplicație și serviciile disponibile într-o rețea sau între două programe de aplicație diferite. În cazul conexiunilor la baze de date Java, produsele middleware pot face sistemul RDBMS să se comporte ca și când ar fi o bază de date orientată pe obiecte și care rulează pe un server situat la distanță. Programatorul Java accesează baza de date folosind metode Java standard, iar produsul middleware se ocupă de translatarea dintre obiecte și componentele bazei de date relaționale. Există în comerț mai multe medii de dezvoltare a aplicațiilor care oferă o soluție middleware între structura de clase Java și bazele de date relaționale. Majoritatea acestor produse pun la dispoziție un instrument grafic pentru maparea datelor relaționale la clasele Java, iar prin folosirea specificației de mapare, generează automat instrucțiunile SQL necesare pentru extragerea datelor din baza de date și popularea claselor, ca și instrucțiunile SQL necesare pentru aplicarea actualizărilor de date în baza de date relațională.

.NET Framework

.NET este strategia pentru servicii web a companiei Microsoft pentru aplicațiile integrate în platforma Microsoft. .NET Framework este un mediu de dezvoltare și de execuție ce permite diferitelor limbaje de programare și biblioteci să lucreze împreună pentru crearea de aplicații. .NET Framework este format din următoarele componente:

- **Common Language Runtime (CLR)** Este un mediu de dezvoltare și de execuție independent de limbaj. Este asigurat suportul pentru limbajele standard, cum sunt C și C++, ca și pentru limbajele proprietare Microsoft, adică Visual Basic .NET, C# și J#.

- **Framework Class Library (FCL)** Este o bibliotecă orientată pe obiecte, solidă, ce conține funcții pregătite.
- **Infrastructura** Reprezintă suportul pentru protocoalele și specificațiile standard de rețea, bibliotecile de programare ale diferitelor limbaje și diferite platforme.

Aplicațiile realizate cu ajutorul mediului de lucru .NET Framework pot folosi clase și biblioteci din cadrul de lucru pentru a trimite instrucțiuni SQL către o bază de date relațională (de obicei Microsoft SQL Server) și pentru a prelucra rezultatele executării instrucțiunilor SQL de către sistemul DBMS. Am inclus .NET Framework aici ca exemplu de mediu de dezvoltare particular unui producător care include resurse pentru înglobarea instrucțiunilor SQL într-o aplicație. Sunt disponibile multe alte medii de acest tip de la o varietate de producători.

Limbajul SQL complet computațional

Așa cum a fost dezvoltat inițial, limbajul SQL era doar un limbaj de date, pentru că îi lipseau o serie de instrucțiuni esențiale în limbajele de programare de tip general, între acestea numărându-se instrucțiunile pentru cicluri, ramificări și tratarea erorilor. Limbajul SQL care include toate instrucțiunile necesare pentru crearea de programe complete este numit *complet computațional*. Atunci când dezvoltatorii și administratorii de baze de date aveau nevoie să efectueze operații complexe cu bazele de date, trebuiau să recurgă la un limbaj de programare de tip general, cum este C. Pentru aplicațiile comerciale (de afaceri), utilizarea unui limbaj de acest tip era (și este în continuare) o soluție potrivită. Însă forțarea administratorilor de baze de date să scrie proceduri de declanșare și proceduri memorate într-un limbaj de programare de tip general (singura opțiune disponibilă în unele sisteme RDBMS timpurii, cum era DB2) nu era o idee prea grozavă, în special pentru că deseori administratorii de baze de date nu cunoșteau foarte bine limbajul de programare suportat.

Procedurile de declanșare (triggers) și *procedurile memorate (stored procedures)* sunt module de program stocate în baza de date. Diferența dintre cele două tipuri este că o procedură de declanșare este executată (inițiată) automat de un eveniment din baza de date, cum ar fi introducerea unui rând într-un anumit tabel, în vreme ce o procedură memorată este executată doar atunci când este apelată printr-o instrucțiune SQL specială, cum ar fi CALL sau EXECUTE. Procedurile memorate sunt utile pentru ascunderea instrucțiunilor SQL complexe de programele de aplicație, de dezvoltatori și de utilizatorii finali ai bazei de date.

De fapt, unii cred că întregul acces la baza de date trebuie să fie efectuat prin proceduri memorate pentru a feri utilizatorii bazei de date de orice modificare a structurii bazei de date. Procedurile de declanșare sunt utile pentru automatizarea unor evenimente, cum sunt înregistrarea modificărilor de date și replicarea (copierea) datelor în altă bază de date,

împreună cu orice altceva trebuie rulat automat pe baza unui eveniment. Este recomandat însă să păstrați structura logică a afacerii în programele de aplicație și nu în baza de date pentru a minimiza cerințele de modificare a bazei de date atunci când se modifică afacerea.

Producătorii de sisteme RDBMS au pus la dispoziție rapid extensii proprietare de limbaj pentru SQL, care au permis scrierea de programe complete. Următoarele două secțiuni din acest capitol prezintă aspecte de ordin general referitoare la două astfel de limbaje: Transact-SQL, oferit împreună cu Microsoft SQL Server și Sybase Adaptive Server (cele două sisteme sunt similare, dar nu au implementări identice) și PL/SQL, oferit împreună cu sistemul RDBMS Oracle. Rețineți că pot fi scrise cărți întregi (și au fost scrise) despre Transact-SQL și PL/SQL; aici sunt doar prezentări succinte.

Transact-SQL (Microsoft SQL Server și Sybase Adaptive Server)

Microsoft SQL Server și Sybase Adaptive Server folosesc același limbaj SQL pentru că au aceleași origini. În 1987, Microsoft a recunoscut că îi lipsește o implementare SQL care să ruleze în noul sistem de operare Windows Server. A fost format un parteneriat cu Sybase, iar rezultatul a fost adaptarea tehnologiei de baze de date a companiei Sybase care rula atât în Windows, cât și în Unix. La sfârșitul colaborării, în 1993, ambele companii au păstrat drepturile asupra tehnologiei dezvoltate, iar rezultatele s-au numit Microsoft SQL Server și Sybase DataServer (denumit ulterior Adaptive Server). În timp, aceste două linii de produse s-au distanțat, însă și în prezent, dacă știți Transact-SQL pentru unul dintre ele, puteți să vă adaptați ușor cunoștințele și să îl învățați pe celălalt.

Transact-SQL include toate instrucțiunile și funcțiile SQL prezentate până în acest moment, incluzând SELECT, INSERT, UPDATE și DELETE, plus toate instrucțiunile necesare pentru a face limbajul complet computațional. În plus, Sybase și Microsoft au pus la dispoziția utilizatorilor un număr considerabil de proceduri memorate, scrise în Transact-SQL, pentru a asista în gestionarea și folosirea bazei de date. Procedurile sunt simplu de recunoscut, pentru că numele lor încep cu „sp_".

În tabelul următor sunt prezentate pe scurt elementele limbajului Transact-SQL:

Element de limbaj	Descriere
BACKUP	Salvează baza de date, jurnalul de tranzacții ori unul sau mai multe fișiere sau grupuri de fișiere.
BEGIN	Începe un bloc de instrucțiuni.
BREAK	Închide un ciclu WHILE.
CASE	Evaluează o listă de condiții și returnează una dintre expresiile rezultat posibile.
CONTINUE	Repornește un ciclu WHILE.

Element de limbaj	Descriere
Instrucțiuni pentru controlul cursorului	Include instrucțiunile DECLARE CURSOR, OPEN, FETCH, CLOSE și DEALLOCATE, care au fost descrise în acest capitol.
Instrucțiuni DCL	Include instrucțiunile SQL standard GRANT și REVOKE, care au fost descrise în capitolul 8.
Instrucțiuni DDL	Include instrucțiunile CREATE, ALTER și DROP, care au fost descrise în capitolul 3, cu extensii pentru crearea și gestionarea procedurilor de declanșare și a procedurilor memorate scrise în Transact-SQL.
DECLARE	Definește variabilele Transact-SQL.
Instrucțiuni DML	Include instrucțiunile SQL standard INSERT, UPDATE și DELETE, care au fost descrise în capitolul 7.
END	Închide un bloc de instrucțiuni.
EXEC	Execută o procedură memorată.
Funcții	Include funcțiile SQL, care au fost descrise în capitolele 4 și 6.
GOTO	Transferă controlul necondiționat (prin ramificare) către o etichetă de instrucțiune.
IF...ELSE	Evaluează una sau mai multe condiții și inițiază instrucțiunile care trebuie executate atunci când o condiție are valoarea TRUE.
PRINT	Returnează un mesaj către client.
RESTORE	Restaurează o bază de date, jurnalul de tranzacții ori unul sau mai multe fișiere sau grupuri de fișiere dintr-o copie de siguranță.
RETURN	Închide procedura curentă, returnând controlul către modulul apelant.
SELECT [INTO]	Include instrucțiunile SQL standard SELECT, care au fost descrise în capitolele de la 4 la 6, cu o clauză opțională INTO adăugată pentru a permite stocarea datelor selectate în variabilele declarate ale limbajului Transact-SQL.
SHUTDOWN	Oprește imediat motorul RDBMS.
Instrucțiuni de tranzacție	Include instrucțiunile BEGIN TRANSACTION, END TRANSACTION, COMMIT și ROLLBACK, care au fost descrise în capitolul 9.
TRUNCATE TABLE	Șterge toate datele dintr-un tabel, însă fără a iniția procedurile de declanșare definite pentru tabel.
USE	Denumește baza de date pe care o vor folosi toate instrucțiunile Transact-SQL ulterioare.
WAITFOR	Întârzie executarea instrucțiunii.
WHILE	Repetă instrucțiunile cât timp o anumită condiție are valoarea TRUE.

Oracle PL/SQL

PL/SQL (Programming Lanuguage/SQL), cunoscut inițial ca „opțiune procedurală", a fost lansat pe piață în 1991 ca parte a sistemului Oracle versiunea 6. A fost modelat după Ada, un limbaj de programare care la momentul respectiv era limbajul standard pentru guvernul Statelor Unite. Poate părea o opțiune ciudată, însă Oracle Corporation a avut întotdeauna legături strânse cu guvernul federal american. De fapt, este bine cunoscut faptul că CIA a fost primul client al companiei Oracle.

Spre deosebire de Transact-SQL, Oracle SQL nu este considerat ca o parte din limbajul PL/SQL, ci un limbaj separat care poate fi apelat din PL/SQL atunci când este necesar. La fel ca Microsoft SQL Server și Sybase Adaptive Server, sistemul Oracle pune la dispoziție foarte multe proceduri memorate împreună cu baza de date, majoritatea scrise în PL/SQL; de obicei numele acestor proceduri încep cu „DBMS_". Combinația dintre SQL, PL/SQL și clientul SQL*Plus (care are propriul set de comenzi) poate fi confuză uneori, deoarece este dificil să vă amintiți cărui limbaj îi aparține o anumită instrucțiune. Însă în afară de găsirea manualului care descrie comanda (atunci când aveți nevoie de sintaxă), nu trebuie să vă îngrijorați prea mult, pentru că SQL*Plus, Oracle SQL și PL/SQL interoperează ca și când ar fi un singur produs.

În tabelul următor sunt prezentate pe scurt elementele limbajului PL/SQL:

Element de limbaj	Descriere
BEGIN	Începe un bloc de instrucțiuni.
CALL	Apelează o procedură PL/SQL ca pe un subprogram.
CURSOR FOR LOOP	Definește un ciclu asociat la un cursor, care este repetat pentru fiecare rând returnat de cursor.
Instrucțiuni pentru controlul cursorului	Include instrucțiunile DECLARE CURSOR, OPEN, FETCH și CLOSE, care au fost descrise în capitolul 10. De asemenea, există variabile speciale (atribute) asociate cu cursoarele pentru a permite operații cum ar fi testarea găsirii unui rând de date de către o instrucțiune FETCH.
DECLARE	Definește variabilele PL/SQL. PL/SQL suportă toate tipurile de date SQL din Oracle, iar în plus adaugă foarte multe tipuri de date proprii, cum ar fi matrice, tabele de memorie, structuri de înregistrări și tipuri definite de utilizator.
END	Închide un bloc de instrucțiuni.
END LOOP	Marchează sfârșitul unei instrucțiuni LOOP sau FOR LOOP.
EXCEPTION	Deschide o secțiune specială a programului PL/SQL care tratează excepțiile (erorile) detectate pe parcursul executării instrucțiunilor. Pentru fiecare condiție care trebuie tratată este inclusă câte o instrucțiune WHEN.

Element de limbaj	Descriere
EXECUTE	Apelează (execută) o procedură memorată PL/SQL.
EXIT [WHEN]	Iese dintr-un ciclu.
FOR LOOP	Definește un ciclu care iterează o variabilă numerică. Poate fi inclus un interval opțional pentru a controla numărul de execuții ale ciclului.
GOTO	Transferă controlul necondiționat (prin ramificare) către o etichetă de instrucțiune.
IF...THEN...ELSE	Evaluează una sau mai multe condiții și inițiază instrucțiunile care trebuie executate atunci când o condiție are valoarea TRUE.
LOOP	Definește un ciclu simplu care va fi executat continuu până va fi oprit de o instrucțiune EXIT.
NULL	Este o instrucțiune care nu face absolut nimic (și îmi amintește de câțiva foști colaboratori). Este utilizată deseori ca indicator într-un bloc de instrucțiuni condiționale.
RAISE	Activează o condiție de eroare. Dacă pentru condiție este definită o instrucțiune WHEN, aceasta preia controlul pentru a trata condiția; în caz contrar, programul PL/SQL eșuează.
RETURN	Închide blocul sau procedura PL/SQL, returnând controlul către blocul sau procedura apelante.
Instrucțiuni de tranzacție	Include instrucțiunile COMMIT, ROLLBACK și SAVEPOINT, care au fost descrise în capitolul 9.
WHEN	Utilizată în secțiunea EXCEPTION, definește o condiție de excepție și acțiunile care trebuie efectuate la apariția excepției. Rețineți că excepțiile pot fi activate atunci când apar erori sau când este executată instrucțiunea RAISE.
WHILE	Repetă instrucțiunile cât timp o anumită condiție are valoarea TRUE.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Un cursor este
   a. Colecția de rânduri returnate de o interogare la baza de date
   b. Un pointer (indicator) într-un set de rezultate

c. Același lucru cu un set de rezultate

d. Un buffer care păstrează rândurile extrase din baza de date

e. O metodă de analiză a performanței instrucțiunilor SQL

2. Un set de rezultate este

a. Colecția de rânduri returnate de o interogare la baza de date

b. Un pointer (indicator) într-un cursor

c. Același lucru cu un cursor

d. Un buffer care păstrează rândurile extrase din baza de date

e. O metodă de analiză a performanței instrucțiunilor SQL

3. Înainte ca rândurile să poată fi extrase dintr-un cursor, cursorul trebuie să fie

a. Declarat

b. Finalizat

c. Deschis

d. Închis

e. Dezalocat

4. Cursoarele sunt

a. Destinate să anuleze nepotrivirea dintre modul în care tratează rezultatele unei interogări limbajele orientate pe obiecte și bazele de date relaționale

b. Destinate să anuleze nepotrivirea dintre modul în care tratează rezultatele unei interogări limbajele procedurale și bazele de date relaționale

c. Incluse în limbajul Oracle PL/SQL

d. Incluse în limbajul Sybase Transact-SQL

e. Incluse în limbajul Microsoft Transact-SQL

5. Numele cursorului este inclus în

a. Instrucțiunea DECLARE CURSOR

b. Instrucțiunea SELECT

c. Instrucțiunea OPEN

d. Instrucțiunea FETCH

e. Instrucțiunea CLOSE

6. Numele cursorului trebuie să fie unic în
   a. Un tabel
   b. Un rând
   c. Un set de rezultate
   d. O bază de date
   e. Un program
7. O instrucțiune OPEN pentru cursor
   a. Determină întotdeauna rularea interogării și completarea setului de rezultate cu date
   b. Poate determina rularea interogării și plasarea unor date în setul de rezultate
   c. Trebuie să includă o clauză INTO
   d. Trebuie să includă numele cursorului
   e. Trebuie să includă numele tabelului (tabelelor)
8. ODBC este
   a. O interfață API standard pentru conectarea la sistemele DBMS
   b. Independent de orice limbaj, sistem de operare sau sistem DBMS
   c. Un standard Microsoft
   d. Utilizat de programele Java
   e. Flexibil în tratarea limbajelor SQL proprietare
9. JDBC este
   a. O interfață API standard pentru conectarea la sistemele DBMS
   b. Independent de orice limbaj, sistem de operare sau sistem DBMS
   c. Un standard Microsoft
   d. Utilizat de programele Java
   e. Flexibil în tratarea limbajelor SQL proprietare
10. JSQL este
    a. Un standard Sun Microsystems
    b. O metodă de înglobare a instrucțiunilor SQL în Java
    c. O extensie a unui standard ISO/ANSI

d. O soluție middleware

e. Independent de orice limbaj, sistem de operare sau sistem DBMS

11. Soluțiile middleware pentru conexiunile Java

a. Folosesc metode Java standard pentru a accesa un sistem RDBMS

b. Fac sistemul RDBMS să arate la fel ca o bază de date orientată pe obiecte

c. Oferă o metodă de înglobare a instrucțiunilor SQL în codul Java

d. Sunt independente de orice limbaj, sistem de operare sau sistem DBMS

e. Rulează de obicei într-un server situat la distanță

12. Microsoft .NET Frameworks include

a. Limbajul Transact-SQL

b. Common Language Runtime (CLR)

c. Common Gateway Interface (CGI)

d. Framework Class Libraries (FCL)

e. Infrastructura de suport pentru diferite specificații de rețea, limbaje de programare și platforme

13. Limbajul SQL complet computațional include instrucțiuni pentru

a. Scrierea de rapoarte

b. Tratarea erorilor

c. Cicluri

d. Ramificări

e. Procesarea cursorului

14. O procedură de declanșare este

a. Executată doar atunci când este apelată

b. Executată automat pe baza unui eveniment din baza de date

c. Scrisă într-un limbaj neprocedural

d. Scrisă într-un limbaj procedural

e. Stocată în baza de date

15. O procedură memorată este
    a. Executată doar atunci când este apelată
    b. Executată automat pe baza unui eveniment din baza de date
    c. Scrisă într-un limbaj neprocedural
    d. Scrisă într-un limbaj procedural
    e. Stocată în baza de date
16. Transact-SQL
    a. Apare în Oracle și Microsoft SQL Server
    b. A apărut prima dată în 1987
    c. Include limbajul SQL standard însoțit de extensii procedurale
    d. A fost dezvoltat în parteneriat de Oracle și Sybase
    e. Este limbajul folosit pentru numeroase proceduri memorate oferite de Microsoft și Sybase
17. Elementele de limbaj incluse în Transact-SQL sunt
    a. BEGIN și END pentru a începe și încheia blocurile de instrucțiuni
    b. WHILE pentru repetarea instrucțiunilor
    c. Instrucțiunile SQL DDL, DQL, DML și DCL
    d. DECLARE pentru definirea variabilelor
    e. GOTO și EXIT pentru ieșirea din cicluri
18. PL/SQL
    a. A fost cunoscut inițial ca Ada
    b. A fost lansat prima dată în 1991
    c. A fost dezvoltat de CIA
    d. Nu include instrucțiuni SQL standard
    e. A apărut prima dată în sistemul Oracle versiunea 6
19. Elementele de limbaj incluse în PL/SQL sunt
    a. DECLARE pentru definirea variabilelor
    b. WAITFOR pentru întârzierea executării instrucțiunilor

c. EXCEPTION pentru tratarea excepțiilor

d. FOR LOOP și WHILE pentru formarea ciclurilor

e. Instrucțiunea SELECT

20. Elementele de limbaj incluse atât în PL/SQL, cât și în Transact-SQL sunt

a. BACKUP și RESTORE pentru salvarea și restaurarea elementelor din baza de date

b. IF...ELSE pentru executarea condițională a instrucțiunilor

c. NULL ca indicator care nu execută nici o acțiune

d. DECLARE pentru definirea variabilelor

e. RETURN pentru ieșirea dintr-un bloc de instrucțiuni

CAPITOLUL 10

Elemente de ajustare și performanță ale limbajului SQL

Acest capitol final prezintă idei de ajustare și performanță pentru rularea mai rapidă și mai eficientă a instrucțiunilor SQL pe care le scrieți. În primul rând, vom arunca o privire asupra soluțiilor generale ce pot fi aplicate la nivelul sistemului DBMS, iar apoi vom studia o serie de indicații pentru scrierea de instrucțiuni SQL eficiente. Numeroasele diferențe dintre sistemele RDBMS (chiar și în cazul diferitelor versiuni ale aceluiași producător) fac dificilă dezvoltarea de soluții universale pentru problemele de performanță. Ceea ce merge bine într-un sistem DBMS are de obicei un efect mai slab sau chiar negativ în alt sistem. Am inclus în acest capitol atât indicații universal aplicabile, cât și unele din cele mai importante indicații specifice producătorilor. Asigurați-vă că ați verificat documentația sistemului RDBMS propriu pentru a afla informațiile curente.

Orice administrator experimentat de baze de date vă poate spune că ajustarea performanțelor bazelor de date este o sarcină continuă. Se pare că mereu există ceva ce poate fi optimizat pentru a funcționa mai bine. Cheia pentru reușită este modul în care vă gestionați timpul, așteptările utilizatorilor bazei de date și setarea cererilor de performanță pentru aplicație chiar înainte de a fi scrisă. Formulările simple, cum ar fi „toate actualizările bazei de date trebuie să se încheie în 4 secunde" sunt de obicei cele mai bune. Odată încheiată această etapă, îmbunătățirea performanțelor devine o chestiune de urmărire a aspectelor care nu corespund cererilor de performanță și de ajustare a lor. Ajustarea unei baze de date respectă regula beneficiilor reduse și este posibil să depuneți mult efort în procesul de optimizare a unei baze de date pentru a obține un câștig mic sau chiar nici un câștig. Interesant în a avea cerințe standard de performanță este faptul că vă puteți opri atunci când procesul atinge nivelul cerințelor și puteți trece la problema următoare.

Considerații generale de ajustare a sistemelor RDBMS

Cele mai multe probleme de performanță a sistemelor RDBMS apar ca rezultat al instrucțiunilor SQL scrise neprofesionist, dar sunt și lucruri pe care administratorul de baze de date le poate face pentru ca întregul sistem de baze de date să ruleze mai eficient. Ca urmare, înainte de a trece la ajustarea interogărilor SQL, secțiunile care urmează oferă câteva idei generale pentru optimizarea sistemelor RDBMS.

Minimizarea numărului de scrieri și citiri pe disc

Poate părea evident, dar cele mai lente operații în majoritatea calculatoarelor sunt operațiile I/O (de intrare/ieșire), cum ar fi operațiile de citire a datelor din sistemul de stocare sau de scriere în sistemul de stocare (hard-discuri). Mutarea datelor în memorie și efectuarea diverselor calcule și transformări ale datelor sunt operații mult mai rapide decât o operație I/O. Atunci, de ce nu puneți întreaga bază de date în memorie pentru a o păstra acolo? De fapt, aceasta chiar s-ar putea întâmpla într-o zi. Totuși, deocamdată, memoria este prea scumpă pentru a înlocui complet hard discurile, iar cea mai mare parte din memorie este *volatilă* , ceea ce înseamnă că datele păstrate în ea se pierd când calculatorul este oprit. Există și memorii nevolatile, însă costul acestora este considerabil mai ridicat. Prin urmare, cel mai bun lucru pe care un administrator de baze de date îl poate face este să folosească eficient memoria pe care o are la dispoziție pentru a minimiza numărul de operații I/O și timpul irosit așteptând ca acestea să fie terminate. Există câteva căi pentru a realiza aceasta:

- **Alocarea de buffere dimensionate corect.** Probabil vă amintiți că un buffer reprezintă o zonă din memorie folosită pentru a păstra date citite recent din sistemul de stocare sau urmează a fi scrise pe acesta. Folosind buffere

dimensionate corespunzător, datele citite recent vor sta în memorie pentru un timp rezonabil, creând posibilitatea ca o interogare nouă să găsească datele de care are nevoie plasate deja în buffer. În ceea ce privește ieșirea, dacă există suficient spațiu în buffer, sistemul RDBMS poate să scrie date în buffer și să copieze conținutul buffer-ului din memorie în sistemul de stocare pentru înregistrarea permanentă în vederea folosirii ulterioare. Aceasta se numește *operație I/O asincronă* și este fundamentală pentru un sistem RDBMS eficient.

- **Dispersarea operațiilor I/O cu discul.** Dacă aveți mai multe unități de disc fizice, dispersați fișierele bazei de date pentru a permite operații I/O paralele. (nu se ia în considerare situația în care aveți mai multe partiții pe un hard-disc al unui calculator personal, deoarece acele partiții sunt plasate pe aceeași unitate de disc fizică). În general, o unitate de disc poate accesa doar o locație din unitate la un moment dat, astfel încât operațiile I/O multiple cu o singură unitate trebuie efectuate serial. Prin dispersarea fișierelor, pot fi efectuate în același timp mai multe operații I/O (câte una cu fiecare unitate).

Ajustarea sistemului de calcul și a mediului de lucru

Este evident că acele calculatoare pe care rulează sistemul DBMS trebuie să funcționeze cât mai rapid și mai eficient posibil. Iată câteva considerații cu caracter general:

- **Alegeți componente hardware rapide și sigure.** Cu cât o componentă hardware din calculator este mai rapidă, în special hard-discurile și procesoarele, cu atât mai rapid vor rula interogările bazei dumneavoastră de date.
- **Ajustarea sistemului de operare.** Deși sunt numeroase diferențe între versiunile de sisteme de operare și variantele oferite de producători, de obicei există posibilitatea îmbunătățirii performanțelor. De exemplu, în Microsoft Windows dimensiunea fișierelor de tip swap poate avea un efect semnificativ asupra performanțelor sistemului de operare. Consultați un manual sau o documentație a sistemului de operare pentru a afla alte elemente specifice.

Proiectarea eficientă a tabelelor

Proiectarea unui tabel relațional poate avea efecte foarte mari asupra performanței. Majoritatea sistemelor RDBMS bine cunoscute au manuale sau documentații specifice care descriu cele mai bune practici de lucru pentru proiectarea de tabele eficiente. Iată câteva indicații generale valabile pentru toate implementările SQL:

- **CHAR sau VARCHAR.** Pentru coloanele de tip caracter care au cel mult 5 caractere, tipul de date CHAR este cel mai potrivit, chiar dacă lungimea datelor variază. Tipul de date VARCHAR este mai indicat pentru coloanele cu lungime

variabilă care au în mod obișnuit o dimensiune mai mare de 5 caractere. Pentru coloanele care conțin întotdeauna date de aceeași lungime, CHAR este o alegere evidentă. Motivul pentru care se utilizează tipul CHAR pentru coloane scurte următorul: coloanele VARCHAR includ 1 până la 3 octeți pentru a păstra datele din coloane și apare o supraîncărcare suplimentară de procesare pentru calculul lungimii atunci când datele coloanei se schimbă.

- **Tipuri de date pentru coloane numerice.** Utilizați cel mai mic tip de date care se potrivește. Se irosește mult spațiu dacă, de exemplu, este utilizat tipul de dată BIGINT atunci când pentru date este potrivit tipul INT sau SMALLINT.
- **Tipuri de date identice pentru coloanele cheie primare și externe.** Uniunile dau rezultate mult mai eficiente dacă tipurile de date ale coloanelor utilizate în predicatul de uniune (de obicei cheia primară dintr-un tabel și cheia externă din alt tabel) au tipuri de date identice. Când tipurile de date sunt diferite, sistemul DBMS trebuie să convertească unul dintre ele astfel încât valorile datelor să poată fi comparate corect, dar o astfel de încărcare conduce la o supraîncărcare nedorită.
- **Atenție la procedurile de declanșare.** Astfel de proceduri plasate în tabelele bazei de date reprezintă uneori metode convenabile și/sau necesare pentru a rezolva probleme specifice cum ar fi impunerea restricțiilor complicate. Operațiile efectuate de procedurile de declanșare fac parte din unitatea de lucru a instrucțiunii SQL care modifică tabelul (instrucțiunea care a inițiat procedura de declanșare) și, ca urmare, încetinește acea instrucțiune SQL.

Optimizarea interogărilor SQL

Aproximativ 80% dintre problemele de performanță ale interogărilor unei baze de date pot fi rezolvate prin ajustarea instrucțiunilor SQL. Totuși, trebuie să înțelegeți cum procesează sistemul DBMS respectiv instrucțiunile SQL pentru a putea ști cum să le adaptați. De exemplu, plasarea instrucțiunilor SQL în interiorul procedurilor memorate poate aduce o îmbunătățire remarcabilă a performanței în Microsoft SQL Server sau Sybase, dar nu și cu Oracle.

Planul de execuție al unei interogări este o descriere a modului în care sistemul RDBMS va procesa o anumită interogare, incluzând folosirea indexurilor, logica uniunilor și costul estimativ al resurselor. Este important să învățați cum să folosiți utilitarul „explain plain" din sistemul DMBS propriu (dacă este disponibil un astfel de plan), pentru că vă va arăta exact modul cum va procesa sistemul DBMS instrucțiunile SQL pe care încercați să le ajustați. Exemple privind modul de obținere a planurilor de execuție pentru bazele de date Microsoft SQL Server și Oracle vor fi prezentate mai târziu în acest capitol.

Secțiunea următoare prezintă câteva sugestii generale privind proiectarea și optimizarea interogărilor SQL, și este urmată de secțiuni care oferă recomandări pentru câteva dintre cele mai populare produse RDBMS disponibile pe piață.

Aspecte generale privitoare la sistemele RDBMS

Această secțiune cuprinde idei de proiectare și de ajustare aplicabile în cele mai multe implementări SQL. Ca întotdeauna, este recomandat să consultați documentația sistemului DBMS folosit pentru a afla care sunt cunoștință posibilitățile oferite.

Cunoașterea optimizatorului

Optimizatorul de interogări este acea componentă software din sistemul RDBMS care analizează o instrucțiune SQL pentru a găsi cea mai bună cale de a o rula. Majoritatea optimizatoarelor moderne sunt *bazate pe cost* (*cost-based*), ceea ce înseamnă că evaluează costurile tuturor căilor posibile de execuție a unei instrucțiuni și apoi o aleg pe cea cu costul cel mai scăzut. O componentă importantă a optimizatoarelor bazate pe cost o reprezintă statisticile obținute din baza de date, cum ar fi numărul de rânduri din fiecare tabel și numărul de valori unice pe care le are fiecare coloană indexată. Unele optimizatoare sunt *bazate de reguli* (*rule-based*), ceea ce înseamnă că aplică un set de norme instrucțiunii SQL în schimbul estimării costului atunci când decid care este cel mai bun mod de execuție. Puține optimizatoare, cum ar fi cel realizat de Oracle, oferă posibilitatea alegerii unei soluții *bazate pe cost* sau *bazate pe reguli*. Iată câteva aspecte privind optimizatoarele de interogări:

- **Ordinea numelor de tabele.** Are ordinea numelor de tabele din clauzele JOIN sau FROM vreo influență în ceea ce privește ordinea în care sunt accesate tabelele atunci când sunt efectuate uniunile? Acesta ar fi mai mult cazul unui optimizator bazat pe reguli. În mod ideal, sistemul DBMS trebuie să acceseze la început tabelul cel mai selectiv (acela care va elimina cel mai mare număr de rânduri din setul de rezultate). De exemplu, Oracle versiunea 7 procesa lista cu numele de tabele de la dreapta la stânga, dar această regulă s-a schimbat începând cu versiunea 8, ordinea fiind în prezent de la stânga la dreapta.
- **Ordinea predicatelor de căutare.** Are ordinea predicatelor din clauza WHERE vreo influență asupra ordinii în care sunt evaluate predicatele? Ideal, cel mai restrictiv predicat (cel care elimină cel mai mare număr de rânduri) trebuie să fie evaluat primul.
- **Lipsa de date statistice.** Dacă este utilizat un optimizator bazat pe cost, ce face acesta atunci când nu sunt colectate date statistice de la unul sau mai multe tabele? Unele optimizatoare, cum ar fi cel realizat de Oracle, comută pe caracteristica bazat pe reguli, în timp ce altele iau în considerare valori predefinite pentru datele statistice cerute ori pur și simplu refuză să utilizeze

indexuri și realizează scanări complete ale tuturor tabelelor. O *scanare completă a tabelului* este operația prin care sistemul DBMS citește fiecare rând din tabel pentru a găsi rândurile dorite, ceea ce poate însemna desigur un dezastru de performanță pentru tabelele cu foarte multe rânduri.

- **Rescrierea interogărilor.** Sunt interogările rescrise în forme mai eficiente de către optimizator? De exemplu, multe optimizatoare rescriu automat subinterogările în uniuni echivalente pentru a simplifica procesarea ulterioară. În unele cazuri, s-ar putea să descoperiți că anumite opțiuni DBMS trebuie activate pentru a permite optimizatorului să rescrie interogările.
- **Combinările definițiilor de vizualizări.** Pentru interogările ce folosesc vizualizări, în ce punct combină sistemul DBMS definiția vizualizării (interogarea care definește vizualizarea) în instrucțiunea SQL inițiată de utilizatorul bazei de date? Aceasta are implicații evidente pentru optimizator – cu cât evaluează mai repede întreaga instrucțiune SQL, cu atât mai inteligentă va fi decizia sa.
- **Alte criterii.** Ce alte criterii pot influența optimizatorul? De exemplu, unele optimizatoare vor prefera indexuri unice în detrimentul celor neunice, iar altele vor favoriza folosirea unui index pentru a ordona rândurile în detrimentul sortării setului de rezultate.

Proiectarea eficientă a interogării

Mulți dezvoltatori de aplicații scriu pur și simplu instrucțiunile SQL fără să se gândească și fără să dea prea mare importanță proiectării acestora în scopul unei procesări eficiente. Aceasta este o capcană în care se poate cădea cu ușurință, pentru că SQL este un limbaj neprocedural care creează falsa impresie că pentru a obține rezultatul corect nu contează felul în care sunt scrise instrucțiunile. Instrucțiunile care nu sunt proiectate pot produce probleme mari de performanță. De exemplu, am revăzut recent o instrucțiune SQL care extrage cam 6 milioane de rânduri de date de fiecare dată când este rulată. Aplicația care a inițiat instrucțiunea SQL solicita doar primul rând găsit. Când acea instrucțiune a fost lansată în execuție, întreaga bază de date a fost blocată până când s-a încheiat procesarea. Programatorul care a rezolvat problema a avut nevoie doar de o oră pentru a transforma interogarea într-una eficientă, dar puțină gândire din partea dezvoltatorului inițial ar fi evitat în întregime criza. Iată câteva aprecieri asupra proiectării interogărilor:

- **Cunoașteți datele.** Când scrieți o instrucțiune SQL, ar trebui să știți câte rânduri sunt în fiecare tabel, cât de selective sunt predicatele WHERE și câte rânduri vă așteptați să conțină setul de rezultate. Cu cât este mai mare numărul de rânduri implicate, cu atât mai mult timp ar trebui să petreceți gândindu-vă cum să scrieți cel mai bine instrucțiunea SQL.

- **Minimizați numărul de rânduri returnate.** Cu cât sunt mai puține rânduri în setul de rezultate, cu atât va rula mai eficient interogarea.
- **Evitați scanarea tabelelor mari.** Pentru tabelele care includ mai mult de 1.000 de rânduri, scanarea tuturor rândurilor din tabel în loc de utilizarea unui index poate fi consumatoare de resurse. Și, firește, cu cât este mai mare un tabel, cu atât mai costisitoare devine scanarea acestuia. Scanările complete ale tabelelor au loc în următoarele situații:
  - Interogarea nu conține o clauză WHERE care să limiteze rândurile.
  - Nici una din coloanele referite în clauza WHERE nu se potrivește cu coloana principală a unui index din tabel.
  - Indexul și datele statistice ale tabelului nu au fost actualizate. Cele mai multe optimizatoare de interogări din sistemul RDBMS folosesc date statistice pentru a evalua indexurile disponibile, iar fără date statistice, scanarea unui tabel poate fi mult mai eficientă decât utilizarea unui index.
  - Cel puțin o coloană din clauza WHERE se potrivește cu prima coloană a unui index disponibil, dar comparația folosită previne folosirea unui index. Aceste cazuri includ următoarele situații:
    - Folosirea operatorului NOT (de exemplu, WHERE NOT CITY = 'New York'). În general, indexurile pot fi folosite pentru a afla ce se găsește într-un tabel, dar nu pot fi folosite pentru a găsi ceea ce nu se află în tabel.
    - Folosirea operatorului NOT EQUAL (de exemplu, WHERE CITY <> 'New York') .
    - Folosirea unui caracter de înlocuire în prima poziție a unui șir de comparație (de exemplu, WHERE CITY LIKE '%York%').
    - Folosirea unei funcții SQL în comparație (de exemplu, WHERE UPPER (CITY)='NEW YORK').
- **Evitați coloanele nefolositoare.** Cu cât datele ocupă mai mult spațiu în fiecare rând din setul de rezultate, cu atât sunt necesare mai multă memorie și mai mult spațiu pe disc pentru operații intermediare, cum ar fi sortările, și pentru păstrarea setului de rezultate.
- **Evitați tabelele nefolositoare.** Cu atât sunt mai puține tabele, cu atât interogarea este mai eficientă.
- **Evitați sortările de seturi mari de rezultate.** Sortările sunt consumatoare de resurse, în special când rândurile sortate nu au loc în memorie. Atunci când vă așteptați ca setul de rezultate să fie foarte mare, trebuie să evitați sortările. Majoritatea optimizatoarelor vor folosi un index dacă acesta poate elimina necesitatea unei sortări, dar crearea unui index doar pentru a evita sortarea unui număr mare de rânduri nu este probabil un lucru inteligent din cauza supraîncărcării necesare păstrării indexului.

- **Potriviți tipurile de date din predicate.** Fie că un predicat compară două valori de coloană (ca în cazul uniunilor) ori compară o valoare a coloanei cu o valoare literală (ca în cazul filtrării rândurilor), este important ca tipul de date să fie același. Când tipurile de date nu corespund, sistemul DBMS trebuie să îl convertească pe unul dintre ele înainte de a face comparația; dacă pentru o operație individuală efortul necesar este relativ mic, acesta este sporit atunci când conversia trebuie făcută pentru fiecare rând dintr-un tabel extins.
- **Folosiți operatorul IN în locul lui OR oricând este posibil.** Operatorul IN poate fi rescris ca JOIN, însă operatorul OR necesită adesea rularea mai multor interogări pentru rezultatele combinate de sistemul DBMS. Primul operator este mult mai eficient.
- **Folosiți GROUP BY în loc de DISTINCT.** În majoritatea sistemelor DBMS, GROUP BY este o metodă mult mai eficientă de eliminare a rândurilor duplicate în comparație cu cuvântul cheie DISTINCT. Motivul este acela că GROUP BY apelează sortarea necesară găsirii duplicatelor mai devreme în procesarea interogării, în timp ce metoda DISTINCT efectuează sortarea în ultima etapă (aplicată setului final de rezultate). Cu cât rândurile duplicate sunt eliminate mai rapid, cu atât mai eficient vor fi prelucrate datele rămase din setul de rezultate.
- **Folosiți indicii dacă este nevoie.** Indiciile reprezintă o sintaxă specială care poate fi introdusă în instrucțiunea SQL pentru a direcționa optimizatorul către efectuarea anumitor acțiuni, cum ar fi impunerea folosirii unui anumit index sau a unei anumite metode pentru a reuni tabelele. Chiar dacă poate fi o opțiune foarte atractivă, ar trebui folosită doar ca o ultimă variantă, deoarece indiciile nu sunt portabile între bazele de date furnizate de producători diferiți și uneori încetează să mai funcționeze atunci când sistemul software DBMS este actualizat la o versiune nouă. Optimizatoarele Oracle, MySQL și Microsoft SQL Server răspund la indicii, dar sintaxa acceptată de fiecare este diferită.
- **Tabelele temporare pot fi folositoare.** Tabelele temporare pot fi utile în unele situații, cum ar fi asamblarea unor seturi mari de rezultate și apoi adăugarea unui index (sau două) pentru a putea susține mai multe interogări ulterioare în tabelele temporare. Totuși, rețineți că dublați citirile și scrierile atunci când procedați astfel, deoarece toate datele selectate din tabelele inițiale (de bază) trebuie scrise în tabelele temporare și apoi recitite de acolo. Pe scurt, totul costă!
- **Vizualizările pot fi folositoare.** Vizualizările pot fi utile deoarece ascund operații complexe, cum ar fi funcțiile de agregare imbricate. Și în sistemele DBMS care nu au zone cache pentru instrucțiunile SQL, vizualizările pot fi procesate mai eficient decât interogările obișnuite, deoarece instrucțiunea SQL care definește vizualizarea a fost deja analizată și optimizată, ceea ce înseamnă că aceste operații nu trebuie efectuate de fiecare dată când este accesată vizualizarea. Dar mai presus de orice, rețineți că vizualizările sunt tot niște

interogări SQL, așa că ele constituie subiectul tuturor opțiunilor de ajustare aplicate oricărei instrucțiuni SQL.

Utilizarea inteligentă a indexurilor

Indexurile pot îmbunătăți semnificativ timpii de acces la date. Totuși, țineți cont întotdeauna de faptul că indexurile ocupă spațiu de stocare și trebuie întreținute. În continuare sunt prezentate o serie de idei pentru îmbunătățirea performanței interogărilor folosind indexurile:

- **Evitați folosirea indexurilor în coloanele actualizate frecvent.** Crearea unui index pentru o coloană ce este actualizată frecvent dublează volumul de scrieri necesare pentru actualizarea coloanei. Nu uitați că atunci când datele coloanei sunt actualizate, sistemul DBMS trebuie să actualizeze toate indexurile care includ acea coloană.
- **Creați doar indexuri selective.** *Selectivitatea indexului* reprezintă un raport între numărul de valori distincte ale unei coloane și numărul de rânduri dintr-un tabel. De exemplu, dacă tabelul conține 1000 de rânduri și o coloană deține 800 de valori distincte, selectivitatea indexului este 0,8, considerată o valoare bună. Însă o coloană care specifică genul și are doar 2 valori distincte (M și F) are o selectivitate foarte slabă (0,002 în acest caz). Indexurile unice au întotdeauna o rată a selectivității de 1,0, cea mai bună posibilă. Una dintre regulile de bază este evitarea indexurilor cu o selectivitate mai mică de 0,33, exceptând cazurile în care acestea sunt indexuri special create pentru selectivitate scăzută, cum ar fi indexurile bit-map.
- **Indexurile de chei externe îmbunătățesc uniunile.** Pentru majoritatea optimizatoarelor, un index pe o coloană cu cheie externă sporește semnificativ performanța și poate permite optimizatorului să utilizeze metode adiționale de uniune.
- **Indexați coloanele utilizate frecvent în predicate.** Pentru tabelele mari, orice interogare ar trebui să conțină un predicat WHERE care să facă referire la o coloană indexată. Prin urmare, cel mai bine este să găsiți coloanele cel mai frecvent referite de predicate și să le indexați.
- **Evitați folosirea în exces a indexurilor.** Ca o regulă de bază, nu creați mai mult de 3 sau 4 indexuri pentru un tabel. După cum am văzut deja, indexurile ocupă spațiu și trebuie întreținute. Prea multe indexuri într-un tabel pot reduce performanța unor comenzi ca INSERT sau UPDATE aplicate pentru acel tabel.
- **Evitați suprapunerea indexurilor.** Aproape orice sistem RDBMS poate utiliza un index chiar și atunci când predicatul WHERE referă doar prima coloană a indexului. Din acest motiv, indexurile suprapuse (cele care au aceeași coloană principală) sunt nefolositoare și redundante.

- **Luați în considerare folosirea indexurilor unice.** În anumite sisteme RDBMS, cum ar DB2, indexurile unice sunt atât de performante, încât administratorii de baze de date adaugă adesea coloane altfel nefolositoare la un index doar pentru a-l face unic.
- **Renunțați la indexuri pentru încărcările mari de date.** Pentru încărcări masive de date într-un tabel luați în considerare ștergerea anumitor indexuri și refacerea acestora după terminarea încărcării. Aceasta poate conduce la un câștig substanțial de timp pentru anumite sisteme DBMS.

Aspecte privind MySQL

În continuare sunt prezentate o serie de soluții de ajustare specifice sistemului DBMS MySQL:

- **Motorul de stocare.** MySQL are o caracteristică unică prin care pune la dispoziție mai multe motoare de stocare; pentru fiecare tabel nou poate fi selectat un singur astfel de motor. Exemple de motoare de stocare sunt MyISAM (un înlocuitor al sistemului ISAM original), HEAP, MERGE, InnoDB și BDB (Berkeley DB). Deosebirile sunt prea complexe pentru a fi explicate aici, dar în manualul de referințe SQL sunt câteva capitole dedicate acestor deosebiri. Este suficient de spus că alegerea motorului de stocare are un efect serios asupra optimizatorului și asupra performanțelor instrucțiunilor SQL aplicate tabelului.
- **Indexurile rezumat.** MySQL suportă indexuri rezumat (hash), în care valorile datelor sunt trimise printr-un algoritm hash, înainte de a fi adăugate indexului. Cu toate că această tehnică dispersează secvențial valorile atribuite astfel încât noile rânduri de date să fie răspândite mai uniform în index, indexurile rezumat au dezavantajul de a nu fi folositoare ca înlocuitoare pentru sortări, deoarece înregistrările indexurilor nu se găsesc în secvența cheie.

Aspecte privind sistemul Oracle

În această secțiune sunt prezentate unele soluții de îmbunătățire specifice bazelor de date Oracle.

Soluții de indexare în Oracle

Oracle furnizează câteva opțiuni suplimentare de indexare, care merită să fie luate în considerație:

- **Indexurile bazate pe funcții.** În mod normal, un predicat de genul WHERE UPPER (MOVIE_TITLE) LIKE 'BIG%' împiedică folosirea unui index. Totuși, Oracle permite crearea unui index printr-o funcție, cum ar fi

UPPER (MOVIE_TITLE), care apoi poate fi folosit de interogările care referă funcția. Sunt câteva condiții obligatorii pentru folosirea acestei opțiuni, așa că s-ar putea să aveți nevoie de ajutor de la administratorul de baze de date Oracle. Sintaxa pentru un index bazat pe funcție folosește specificația funcției în loc de numele coloanei în clauza ON a instrucțiunii CREATE INDEX:

```
CREATE INDEX IX_MOVIE_TITLE_UPPER
 ON MOVIE (UPPER(MOVIE_TITLE));
```

- **Indexurile bit-map.** Indexurile bit-map Oracle sunt create pentru lucrul cu coloane pentru care cardinalitatea este scăzută (adică unde sunt relativ puține valori de date printre multe rânduri). Un *index bit-map* conține înregistrări care au un bit pentru fiecare rând în tabelul indexat și câte o înregistrare pentru fiecare valoare de date posibilă din coloana indexată. Un bit este activ („on" sau 1 binar) dacă rândul corespondent din tabel conține valoarea reprezentată de înregistrarea din index și inactiv („off" sau 0 binar) în caz contrar. De exemplu, dacă tabelul MOVIE conține 5.000 de rânduri și creați un index bit-map pentru coloana MPAA_RATING_CODE, indexul ar conține înregistrări ce ar avea 5.000 de biți (aproximativ 625 de octeți) ca mărime, și ar fi 6 astfel de înregistrări (câte una pentru fiecare valoare a coloanei MPAA_RATING_CODE). Prima înregistrare a indexului ar reprezenta prima valoare MPAA_RATING_CODE („G") și biții din acea înregistrare ar fi „on" când rândul corespondent din tabelul MOVIE are valoarea „G". Sistemul DBMS poate folosi algebra matriceală pentru a găsi rapid rândurile de date dorite, în special când predicatele căutării referă mai multe coloane cu indexuri bit-map. Un index bit-map este creat cu ajutorul sintaxei normale CREATE INDEX în care este folosit cuvântul cheie BITMAP în locul cuvântului cheie UNIQUE:

```
CREATE BITMAP INDEX IX_MOVIE_MPAA_RATING
 ON MOVIE (MPAA_RATING_CODE);
```

- **Tabelele organizate după index.** Este recomandat să creați un index pentru cheia primară a fiecărui tabel. Totuși, pentru tabelele cu puține coloane, cum sunt tabelele de referință și tabelele de intersecție mici, acest lucru pare întotdeauna nefolositor, deoarece majoritatea sau toate datele din tabel sunt repetate în index. Oracle furnizează o soluție bună care permite stocarea întregului tabel în index. În principiu, un tabel organizat în funcție de index (IOT) este un index de cheie primară și un tabel într-o singură structură. Deși puteți să creați indexuri suplimentare într-o structură IOT, este recomandat să folosiți un tabel convențional. În continuare este prezentat un exemplu care creează un tabel de referință pentru formate media cum este oo structură IOT:

```
CREATE TABLE MEDIA_FORMAT (
  MEDIA_FORMAT_CODE CHAR(1),
  MEDIA_FORMAT_DESC VARCHAR(50),
```

```
    CONSTRAINT PK_MEDIA_FORMAT
        PRIMARY KEY (MEDIA_FORMAT_CODE))
    ORGANIZATION INDEX;
```

Folosirea instrucțiunii EXPLAIN PLAN

În Oracle, instrucțiunea SQL EXPLAIN PLAN analizează o instrucțiune SQL și afișează rezultatele analizei într-un tabel special de planuri. Tabelul de planuri trebuie creat exact așa cum a fost specificat de Oracle, astfel că cel mai bine este să folosiți scriptul furnizat în acest scop, care poate fi găsit în directorul de bază Oracle, cu numele /RDBMS/ADMIN/catplan.sql. După rularea instrucțiunii EXPLAIN PLAN, trebuie să extrageți rezultatele din tabelul de planuri folosind instrucțiunea SELECT. Din fericire, Oracle Enterprise Manager (Managerul de proiecte Oracle) are disponibilă o versiune grafică (GUI), care simplifică mult operațiile de ajustare a interogărilor.

În continuare este prezentat un exemplu de instrucțiune EXPLAIN PLAN:

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'STMT_1'
 FOR
  SELECT MOVIE_ID,
         MOVIE_GENRE.MOVIE_GENRE_DESCRIPTION AS GENRE,
         MOVIE.MPAA_RATING_CODE AS RATING,
         MPAA_RATING.MPAA_RATING_DESCRIPTION AS RATING_DESC
    FROM MOVIE
         JOIN MOVIE_GENRE ON MOVIE.MOVIE_GENRE_CODE =
                             MOVIE_GENRE.MOVIE_GENRE_CODE
         JOIN MPAA_RATING ON MOVIE.MPAA_RATING_CODE =
                             MPAA_RATING.MPAA_RATING_CODE
   WHERE MOVIE_ID < 6
   ORDER BY MOVIE_ID;

Explained.
```

NOTĂ:

- *STATEMENT_ID este orice șir de caractere pe care autorul instrucțiunii dorește să îl folosească pentru a identifica rezultatele explicate. Această caracteristică permite rularea mai multor explicații cu identificatorul STATEMENT_ID ce reține informațiile despre fiecare plan de execuție, separat, în tabelul de planuri.*
- *Instrucțiunea care va fi explicată urmează cuvântului cheie FOR și poate fi orice instrucțiune SQL validă*

- *Atunci când este rulată instrucțiunea EXPLAIN PLAN, instrucțiunea SQL nu este executată. În locul unui set de rezultate ce conține rânduri de date, este returnat utilizatorului doar mesajul „Explained". Acest lucru indică faptul că informațiile privind planul explicativ au fost scrise cu succes în tabelul de planuri.*

Următoarea este instrucțiunea folosită de obicei pentru a extrage și a afișa planul de execuție. Este o instrucțiune SQL complexă, dar singurul lucru care trebuie schimbat la rulare este identificatorul STATEMENT_ID. Clauza CONNECTED BY este o extensie SQL proprie sistemului Oracle, care leagă o relație recursivă într-un tabel prin toate iterațiile (de la copil la părinte, la bunic și așa mai departe). Această instrucțiune SQL a fost inclusă pentru a ilustra o metodă de vizualizare a rezultatelor planului explicativ. După cum a fost menționat anterior, există și un instrument grafic în Oracle Enterprise Manager.

```
SELECT rtrim(substr(LPAD(' ',2*(LEVEL-1))||operation,1,30))||' '
             ||rtrim(options)||' '||rtrim(object_name)|| ' '
             ||'(cost= '||cost||', cardinality='||cardinality||')'
          "Query Plan"
  FROM plan_table
 START WITH id = 0
        AND upper(statement_id) = upper('STMT_1')
CONNECT BY PRIOR id = parent_id
        AND upper(statement_id) = upper('STMT_1');

Query Plan
----------------------------------------------
SELECT STATEMENT (cost= 10, cardinality=5)
   SORT ORDER BY (cost= 10, cardinality=5)
      HASH JOIN (cost= 9, cardinality=5)
         MERGE JOIN (cost= 5, cardinality=5)
           TABLE ACCESS BY INDEX ROWID MPAA_RATING
                        (cost= 2, cardinality=6)
             INDEX FULL SCAN SYS_C005440
                        (cost= 1, cardinality=6)
           SORT JOIN (cost= 3, cardinality=5)
             TABLE ACCESS BY INDEX ROWID MOVIE
                          (cost= 2, cardinality=5)
               INDEX RANGE SCAN SYS_C005449
                          (cost= 1, cardinality=5)
         TABLE ACCESS FULL MOVIE_GENRE
                          (cost= 3, cardinality=16)

10 rows selected.
```

În continuare sunt prezentate câteva puncte cheie privitoare la planul de interogare care apare în setul de rezultate:

- Indentarea indică ordinea de execuție, etapele ilustrate cel mai în interior fiind efectuate primele.
- Valorile de cost indică un cost relativ. Numerele nu au nici o semnificație în afară de diferențele relative. De exemplu, un cost cu valoarea 10 reprezintă o etapă care folosește de două ori mai multe resurse decât o etapă care are costul 5.
- Valorile de cardinalitate indică numărul estimat de rânduri procesate în etapa respectivă.
- „TABLE ACCESS FULL" indică o scanare completă a tabelului, în care toate rândurile din tabel sunt citite succesiv.
- „INDEX RANGE SCAN" indică scanarea unei părți din rândurile unui index.
- „TABLE ACCESS BY INDEX" indică accesul la un tabel folosind indexul indicat.
- Sortările și uniunile ar trebui evidențiate de numele etapelor.

Aspecte privind Microsoft SQL Server

Această secțiune prezintă câteva aspecte de ajustare specifice bazelor de date Microsoft SQL Server.

Aspecte privind performanțele interogărilor SQL Server

În continuare sunt prezentate câteva aspecte privind rularea interogărilor în bazele de date Microsoft SQL Server:

- **Accesul prin intermediul procedurilor memorate.** Amplasarea interogărilor SQL folosite frecvent în proceduri memorate poate conduce la îmbunătățirea semnificativă a performanței. De fapt, în versiunile mai vechi ale sistemului Microsoft SQL Server, era esențial să procedați astfel pentru a obține performanțe rezonabile, deoarece nu existau zone cache SQL care să rețină instrucțiunile SQL recent folosite. În SQL Server 2005, zona cache SQL permite sistemului DBMS să recunoască o instrucțiune similară cu cea care a fost deja rulată și să treacă de etapele de pregătire a instrucțiunii și de optimizare atunci când instrucțiunea este refolosită din zona cache. Cu toate acestea, orice instrucțiune accesată prin intermediul unei proceduri memorate este precompilată si optimizată, astfel încât va funcționa corect întotdeauna (chiar dacă nu se află în zona cache în momentul rulării). Acesta este un motiv pentru care dezvoltatorii experimentați se străduiesc să acceseze în majoritatea situațiilor baza de date SQL Server folosind proceduri memorate.

- **Evitați valorile nule în indexurile unice.** Spre deosebire de majoritatea celorlalte implementări SQL, indexurile unice din SQL Server nu manevrează corect valorile coloanelor nule. Puteți defini un index unic ce include o coloană care poate fi nulă, dar a doua înregistrare din index care conține o valoare nulă este considerată a fi un duplicat al primei înregistrări și conduce la o eroare de cheie duplicată. Majoritatea celorlalte implementări SQL evită această problemă prin neindexarea valorilor nule.
- **Luați în considerare indexurile organizate pe grupuri.** Un index organizat pe grupuri determină ordonarea fizică a rândurilor tabelului în funcție de coloanele indexate. Acest lucru poate fi util în operațiile de căutare și de reuniune, dar în detrimentul întreținerii tabelului, în special atunci când rândurile noi forțează mutarea rândurilor existente pentru a menține secvențialitatea.

Afișarea planurilor de execuție folosind instrumentul SQL Query Analyzer

Microsoft furnizează o facilitate pentru explicarea planurilor de execuție a interogărilor, la fel ca Oracle și DB2. În Microsoft SQL Server 2000, instrumentul SQL Query Analyzer pune la dispoziție un buton numit „Display Estimated Execution Plan" (Afișează planul de execuție estimat), care afișează grafic modul în care va fi rulată instrucțiunea SQL. Această caracteristică poate fi accesată și din meniul Query folosind opțiunea Show Execution Plan (Afișează planul de execuție). Denumirile caracteristicilor pot diferi în alte versiuni Microsoft SQL Server. În SQL Server 2005, nu pare să existe nici o caracteristică numită SQL Query Analyzer și nu se știe cum va fi denumit acest instrument la lansarea produsului.

În instrumentul Query Analyzer, modul prestabilit este Object Browser. Pentru a afișa un plan de execuție, executați clic pe pictograma Display Estimated Execution Plan sau apăsați CTRL+L. Planul de execuție este afișat grafic, așa cum este ilustrat în figura 11-1. Fiecare pas este afișat ca o pictogramă, iar numărul de lângă pictogramă indică procentajul din costul total al interogării caracteristic acelui pas. Puteți poziționa cursorul pe o pictogramă pentru a vedea pe ecran o fereastră pop-up ce prezintă informații detaliate despre acel pas.

Aspecte privind sistemul DB2 UDB

La fel ca Oracle și Microsoft SQL Server, și sistemul DB2 furnizează un utilitar pentru afișarea planurilor de rulare a interogărilor. IBM pune la dispoziție un script numit EXPLAIN.DDL, care creează tabelele necesare pentru a reține rezultatul facilității explicative. După crearea tabelelor explicative, facilitatea de explicare poate fi folosită pentru crearea planurilor de rulare a interogărilor; după aceea, planurile pot fi vizualizate

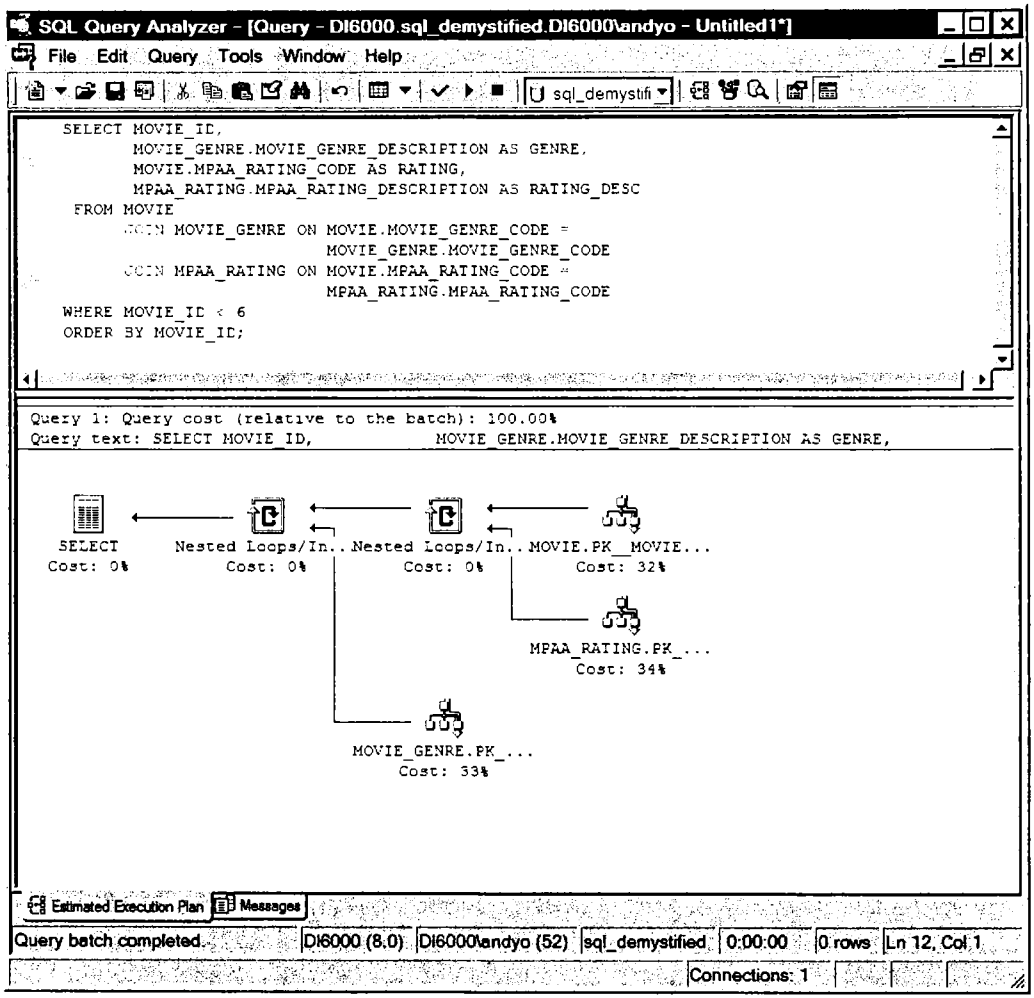


Figura 11-1 Rezultatul returnat de instrumentul Query Analyzer

fie folosind opțiunea Visual Explain din Control Center, fie interogând tabelele explicative cu ajutorul unor instrucțiuni SQL.

Optimizarea instrucțiunilor DML

Instrucțiunile limbajului de manipulare a datelor (Data Manipulation Language sau DML) produc în general mai puține probleme legate de performanță decât instrucțiunile interogative. Cu toate acestea, pot apărea probleme.

Pentru instrucțiunile INSERT, există două aspecte principale:

- **Asigurarea de spațiu liber adecvat pentru rândurile noi.** Câmpurile tabelelor care duc lipsă de spațiu prezintă probleme atunci când sistemul DBMS caută spațiu liber pentru păstrarea rândurilor noi inserate. În plus, inserările nu pun rândurile în tabel în secvența cheii primare, deoarece de obicei nu există spațiu liber în locurile corespunzătoare. De aceea, reorganizarea tabelului, care este în esență un proces de descărcare a rândurilor într-un fișier simplu (plat), re-crearea tabelului și apoi reîncărcarea tabelului pot spori performanțele de inserare și interogare.

- **Întreținerea indexurilor.** De fiecare dată când un rând este inserat într-un tabel, o înregistrare corespondentă trebuie inserată în fiecare index construit pe baza tabelului (cu toate că de obicei valorile nule nu sunt indexate). Cu cât sunt mai multe indexuri, cu atât vor fi necesare mai multe resurse pentru fiecare inserare individuală. Spațiul liber din index poate fi optimizat de obicei la fel ca spațiul liber din tabele.

Pentru instrucțiunile UPDATE există următoarele probleme:

- **Întreținerea indexurilor.** Dacă sunt actualizate coloanele indexate, trebuie actualizate și înregistrările corespunzătoare indexurilor. În general, actualizarea valorilor cheii primare are implicații negative asupra performanței, astfel încât unele sisteme RDBMS interzic operațiile de acest fel.
- **Expansiunea rândurilor.** Atunci când coloanele sunt actualizate în așa fel încât rândul crește semnificativ în mărime, rândul ar putea să nu mai încapă în locația originală și s-ar putea să nu mai existe spațiu liber în jurul lui pentru extindere (este posibil ca alte rânduri să fie exact lângă cel actualizat). Când se petrece acest lucru, rândul trebuie fie să fie mutat în altă locație din fișierul de date, unde va avea loc, fie să fie împărțit, iar partea suplimentară să fie plasată într-o locație nouă, conectată la locația originală cu ajutorul unui pointer. Ambele situații sunt și costisitoare, dar și dăunătoare pentru performanțele interogărilor care extrag acele rânduri. Operațiile de reorganizare a tabelului pot rezolva această problemă, dar este mai bine să o prevenim prin proiectarea aplicației în așa fel încât rândurile să tindă să nu crească în mărime după ce au fost inserate.

Instrucțiunile DELETE generează cele mai puține probleme legate de performanță. Totuși, un tabel care participă ca „părinte" într-o relație definită cu opțiunea ON DELETE CASCADE poate funcționa prost dacă sunt multe rânduri „copii" de șters. Mai mult decât atât, intră în calcul întreținerea indexurilor, deoarece trebuie îndepărtate înregistrările din indexuri pentru orice rând șters.

Întrebări recapitulative

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. Cerințele pentru performanță
   a. Ar trebui stabilite după ajustarea instrucțiunilor SQL
   b. Asigură o cale de identificare a instrucțiunilor ce necesită ajustare
   c. Sunt mai bune atunci când conțin criterii complexe
   d. Asigură o cale de a afla când să se încheie particularizarea unei interogări
   e. Sunt dezvoltate doar pentru a-i face pe auditori fericiți
2. Citirile și scrierile efectuate pe disc pot fi diminuate
   a. Alocând buffere de capacitate suficientă
   b. Plasând toate fișierele bazei de date pe o singură unitate de disc
   c. Introducând toată baza de date în memorie
   d. Adăugând indexuri pentru toate coloanele importante din tabel
   e. Dispersând fișierele pe toate unitățile de disc disponibile
3. Sistemul calculatorului poate fi ajustat
   a. Colectând date statistice ale bazei de date
   b. Aplicând actualizările de securitate disponibile
   c. Selectând componente hardware rapide și fiabile
   d. Respectând recomandările de ajustare ale sistemului SGBD
   e. Consultând un ghid de ajustare al sistemului de operare
4. Un proiect eficient de tabel include
   a. Folosirea tipului VARCHAR pentru toate de tip caracter cu lungime variabilă
   b. Folosirea celui mai mic tip de dată numerică posibil care reține valorile datelor
   c. Folosind proceduri de declanșare oricând este posibil
   d. Folosind tipuri de date identice pentru cheile primare și pentru cheile externe corespunzătoare
   e. Folosind tipuri de date identice pentru toate coloanele cu cheie primară

5. Planul de execuție a unei interogări
   a. Descrie modul în care va rula sistemul DBMS o interogare
   b. Este stocat în zona cache SQL
   c. Este creat folosind utilitarul pentru planuri explicative al sistemului RDBMS
   d. Necesită un tabel de planuri pentru stocarea rezultatelor explicative
   e. Necesită folosirea unei procedurii memorate
6. Optimizatorul de interogări
   a. Creează un plan de execuție a interogării în tabelul de planuri
   b. Determină cea mai bună cale de execuție a unei instrucțiuni SQL
   c. Poate folosi date statistice adunate din baza de date
   d. Poate folosi reguli aplicate modului în care a fost scrisă instrucțiunea
   e. Poate fi bazat pe cost sau bazat pe reguli
7. Pentru a scrie interogări proiectate corect, dezvoltatorul trebuie
   a. Să știe caracteristicile datelor din baza de date
   b. Să potrivească tipurile de date în predicate
   c. Să folosească indicii cât mai des posibil
   d. Să evite coloanele și tabelele care nu sunt necesare
   e. Să maximizeze numărul de rânduri din fiecare set de rezultate
8. Scanările unui tabel pot fi evitate
   a. Incluzând o clauză WHERE care referă o coloană indexată
   b. Folosind GROUP BY în loc de DISTINCT
   c. Asigurându-vă că datele statistice sunt actualizate
   d. Configurând cel puțin un predicat să facă referire la coloana principală a unui index
   e. Evitând coloanele care nu sunt folositoare
9. Un index nu poate fi folosit când
   a. Clauza WHERE face referire la a doua coloană a unui index
   b. Există o clauză LIKE care face referire la un șir de comparație care conține un caracter de înlocuire (cu excepția primei poziții a șirului)

c. Operatorul NOT este folosit într-un predicat

d. O funcție SQL este inclusă într-o comparație de coloane (exceptând cazul în care corespunde unui index bazat pe funcții)

e. Operatorul NOT EQUAL este folosit într-un predicat

10. Aspectele privind folosirea indexurilor includ

a. Amplasarea de indexuri pentru toate coloanele actualizate frecvent

b. Amplasarea de indexuri pentru coloanele cu cheie externă

c. Evitarea suprapunerii indexurilor

d. Crearea de indexuri pentru coloanele care au doar câteva valori posibile

e. Evitarea indexurilor unice

11. Aspectele de ajustare privitoare la MySQL includ

a. Indexuri bazate pe funcții

b. Indexuri rezumat

c. Indexuri organizate pe grupuri

d. Opțiunile motorului de stocare

e. Indexuri bit-map

12. Aspectele de ajustare privitoare la Oracle includ

a. Indexuri bazate pe funcții

b. Indexuri rezumat

c. Indexuri organizate pe grupuri

d. Opțiunile motorului de stocare

e. Indexuri bit-map

13. Aspectele de ajustare privitoare la Microsoft SQL Server includ

a. Indexuri bazate pe funcții

b. Indexuri rezumat

c. Indexuri organizate pe grupuri

d. Opțiunile motorului de stocare

e. Indexuri bit-map

14. Un plan explicativ în Oracle
    a. Necesită folosirea unui tabel de planuri
    b. Conține o coloană PLAN_ID pentru o identificare unică
    c. Poate fi vizualizat folosind instrumentul Enterprise Manager
    d. Poate fi vizualizat folosind limbajul SQL prin selectarea din tabelul de planuri
    e. Este creat folosind instrucțiunea CREATE PLAN
15. Un plan de execuție în Microsoft SQL Server
    a. Necesită folosirea unui tabel de planuri
    b. Poate fi afișat folosind o opțiune din utilitarul SQL Query Analyzer
    c. Afișează planul de execuție într-un format text
    d. Afișează planul de execuție într-un format grafic
    e. Poate fi vizualizat în SQL prin selectarea din tabelul de planuri
16. La ajustarea instrucțiunii INSERT, ar trebui să fie luată în considerație
    a. Întreținerea indexului
    b. Expansiunea rândului
    c. Opțiunea CASCADE
    d. Spațiul liber adecvat
    e. Rescrierile interogărilor
17. La ajustarea instrucțiunii UPDATE, ar trebui să fie luată în considerație
    a. Întreținerea indexului
    b. Expansiunea rândului
    c. Opțiunea CASCADE
    d. Spațiul liber adecvat
    e. Rescrierile interogărilor
18. La ajustarea instrucțiunii DELETE, ar trebui să fie luată în considerație
    a. Întreținerea indexului
    b. Expansiunea rândului
    c. Opțiunea CASCADE

d. Spațiul liber adecvat

e. Rescrierile interogărilor

19. Tipurile de date ar trebui să corespundă

a. Pentru toate coloanele cu cheie primară

b. Între cheia primară și coloanele corespunzătoare cu cheie secundară

c. Între cheia primară și coloanele corespunzătoare cu cheie externă

d. Între valorile din coloană și valorile literale comparate în predicate

e. Pentru toate indexurile bazate pe funcții

20. Principala cauză a problemelor legate de performanțele interogărilor este

a. Un sistem de operare slab ajustat

b. O instrucțiune SQL redactată neprofesionist

c. Supraîncărcarea cu proceduri de declanșare

d. Supraîncărcarea întreținerii indexurilor

e. Expansiunea rândului

Examen final

Alegeți răspunsurile corecte pentru fiecare din următoarele întrebări cu răspunsuri multiple. Rețineți că întrebările pot avea mai multe răspunsuri corecte.

1. SQL este
   a. Un limbaj folosit pentru definirea diagramelor ERD
   b. Folosit pentru definirea paginilor web
   c. Un limbaj inventat de Dr. E. F. Codd
   d. Un limbaj folosit pentru comunicarea cu bazele de date relaționale
   e. Folosit pentru definirea și modificarea obiectelor unei baze de date
2. O bază de date este
   a. Un produs software furnizat de un producător de baze de date
   b. O colecție de date interrelaționate gestionate ca o singură unitate
   c. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index
   d. Definită în același mod de toți producătorii de software
   e. Implementată diferit de diferiți producători

3. Un obiect al unei baze de date este
   a. Un produs software furnizat de un producător de baze de date
   b. O colecție de înregistrări înrudite, stocate ca o singură unitate
   c. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index
   d. Definit în același mod de toți producătorii de software
   e. O colecție de date interrelaționate gestionate ca o singură unitate
4. Un sistem de gestionare a bazelor de date este
   a. Un produs software furnizat de un producător de baze de date
   b. O colecție de date interrelaționate gestionate ca o singură unitate
   c. Deseori abreviat sub forma DBMS
   d. O structură, cum ar fi un tabel, o vizualizare sau un index
   e. O structură denumită de date, cum ar fi un tabel, o vizualizare sau un index
5. Serviciile de bază oferite de un sistem DBMS includ
   a. Mutarea datelor în și din fișiere de date, după cum este necesar
   b. Stocarea datelor într-o formă tabelară
   c. Mecanisme de securitate, pentru a împiedica accesul și modificarea neautorizată a datelor
   d. Generarea diagramelor ERD (Entity Relationship Diagram)
   e. Acceptarea unui limbaj de interogare
6. Componentele unei baze de date relaționale includ
   a. Diagrame ERD
   b. Indexuri
   c. Tabele
   d. Relații
   e. Restricții
7. Într-o diagramă ERD, cardinalitatea maximă este reprezentată prin
   a. Notația „<pk>“ deasupra liniei care reprezintă relația
   b. Un simbol numit „picior de cioară“ desenat la capătul liniei care reprezintă relația
   c. Un cerc desenat pe linia care reprezintă relația
   d. O linie verticală care taie linia care reprezintă relația
   e. Absența unui simbol la sfârșitul liniei care reprezintă relația

8. Printre tipurile de restricții care pot fi folosite în bazele de date se numără
   a. NOT NULL
   b. Unicitate
   c. Cheie primară
   d. CHECK
   e. Relație
9. Scopul normalizării este de a rezolva următoarele probleme
   a. Performanțe reduse
   b. Anomalia de creare
   c. Anomalia de inserare
   d. Anomalia de ștergere
   e. Anomalia de actualizare
10. Pentru a fi în a treia formă normală, o relație
    a. Trebuie să aibă un identificator unic
    b. Trebuie să nu aibă dependențe tranzitive
    c. Trebuie să nu aibă grupuri repetitive sau atribute multivaloare
    d. Trebuie să fie în prima formă normală
    e. Trebuie să fie în a doua formă normală
11. SQL
    a. Poate fi pronunțat ca și cuvântul englezesc „sequel"
    b. Poate fi pronunțat ca literele SQL (es-q-el)
    c. Poate fi folosit pentru comunicarea cu orice bază de date
    d. Poate fi folosit pentru comunicarea cu bazele de date relaționale
    e. Poate fi folosit pentru redarea paginilor web
12. SQL este
    a. Un limbaj standard
    b. Un limbaj neprocedural
    c. Un limbaj procedural
    d. Un limbaj orientat spre obiecte
    e. Un limbaj declarativ

13. Instrucțiunile SQL
    a. Încep cu un delimitator, cum ar fi caracterul punct și virgulă
    b. Se termină un delimitator, cum ar fi caracterul punct și virgulă
    c. Încep cu o paranteză deschisă
    d. Încep cu un cuvânt cheie reprezentând o comandă
    e. Se termină cu un cuvânt cheie reprezentând o comandă
14. Elementele limbajului SQL includ
    a. Cuvinte cheie
    b. Nume ale obiectelor din baza de date
    c. Restricții
    d. Constante
    e. Operatori
15. Instrucțiunile SQL pot fi împărțite în următoarele categorii
    a. Limbajul de definire a datelor (DDL – Data Definition Language)
    b. Limbajul pentru controlul datelor (DCL – Data Control Language)
    c. Limbajul de manipulare a datelor (DML – Data Manipulation Language)
    d. Limbajul de selectare a datelor (DSL – Data Selection Language)
    e. Limbajul de replicare a datelor (DRL – Data Replication Language)
16. Limbajul de definire a datelor (DDL – Data Definition Language) include următoarele instrucțiuni:
    a. CREATE
    b. ALTER
    c. SELECT
    d. UPDATE
    e. DELETE
17. Limbajul de interogare a datelor (DQL – Data Query Language) include următoarele instrucțiuni:
    a. CREATE
    b. ALTER
    c. SELECT
    d. UPDATE
    e. DELETE

18. Limbajul de manipulare a datelor (DML – Data Manipulation Language) include următoarele instrucțiuni:
    a. CREATE
    b. ALTER
    c. SELECT
    d. UPDATE
    e. DELETE
19. Limbajul SQL a fost dezvoltat inițial
    a. De IBM
    b. De ANSI
    c. În anii '80
    d. În anii '70
    e. Pe baza standardelor ANSI
20. Tipurile numerice standard includ
    a. BOOLEAN
    b. INTERVAL
    c. NUMBER
    d. INTEGER
    e. FLOAT
21. Tipurile de date temporale standard includ
    a. TIMESTAMP
    b. TIMEZONE
    c. TIME
    d. DATETIME
    e. DATE
22. Valorile NULL
    a. Sunt egale cu alte valori NULL
    b. Nu sunt egale cu alte valori NULL
    c. Înseamnă același lucru ca și spațiile libere
    d. Pot fi folosite pentru reprezentarea datelor care lipsesc sau nu sunt cunoscute
    e. Sunt întotdeauna permise în mod prestabilit

23. Definiția unei coloane din instrucțiunea CREATE TABLE poate include
    a. O clauză DEFAULT
    b. O clauză NULL sau NOT NULL
    c. Numele tabelului
    d. O restricție la nivel de tabel
    e. Numele coloanei
24. Numele unei coloane dintr-un tabel
    a. Trebuie să fie unic în cadrul bazei de date
    b. Trebuie să fie unic în cadrul tabelului
    c. Trebuie să fie specificat în instrucțiunea CREATE TABLE
    d. Trebuie să fie specificat în instrucțiunea ALTER TABLE
    e. Poate fi folosit într-un singur index
25. O restricție la nivel de coloană
    a. Poate fi folosită oriunde ar putea fi folosită o restricție la nivel de tabel
    b. Poate referi una sau mai multe coloane
    c. Folosește o sintaxă identică sau aproape identică cu cea a unei restricții de același tip la nivel de tabel
    d. Poate fi inclusă într-o instrucțiune CREATE TABLE sau ALTER TABLE
    e. Are o sintaxă care diferă de la un tip de restricție la altul
26. Sintaxa corectă pentru clauza DEFAULT este
    a. DEFAULT [UNIQUE | PRIMARY KEY]
    b. DEFAULT [NULL | NOT NULL]
    c. DEFAULT (*precizie, scală*)
    d. DEFAULT (*expresie*)
    e. DEFAULT (*nume_coloană*) REFERENCES *nume_tabel* (*nume_coloană*)
27. Sintaxa corectă pentru o restricție NOT NULL este
    a. *nume_coloană* REFERENCES NOT NULL
    b. *nume_coloană tip_de_date* NOT NULL
    c. *nume_coloană tip_de_date* IS NOT NULL
    d. CREATE NOT NULL INDEX ON *nume_coloană*
    e. DEFAULT [NULL | NOT NULL]

28. Sintaxa corectă pentru o restricție UNIQUE este
    a. DEFAULT UNIQUE (*nume_coloană*)
    b. *nume_coloană* REFERENCES UNIQUE *nume_tabel*
    c. [CONSTRAINT *nume_restricție*] UNIQUE (*nume_coloană*)
    d. [CONSTRAINT *nume_restricție*] UNIQUE (*nume_tabel*)
    e. DEFAULT [UNIQUE | PRIMARY KEY]
29. Sintaxa corectă pentru o restricție referențială este
    a. REFERENCES *nume_tabel* (*nume_coloană*)
    b. *nume_coloană* REFERENCES UNIQUE *nume_tabel*
    c. [CONSTRAINT *nume_restricție*] REFERENCES *nume_index*
    d. [CONSTRAINT *nume_restricție*] REFERENCES *nume_tabel*
    e. FOREIGN KEY *nume_coloană* REFERENCES *nume_tabel* (*nume_coloană*)
30. Instrucțiunea DROP poate fi folosită pentru a șterge
    a. O coloană dintr-un tabel
    b. Un tabel
    c. O restricție referențială
    d. Un index
    e. O vizualizare
31. Operatorul UNION
    a. Este numit JOIN în unele implementări SQL
    b. Include rândurile duplicate în setul de rezultate
    c. Elimină rândurile duplicate din setul de rezultate
    d. Combină două interogări într-o singură interogare de tip join
    e. Combină seturile de rezultate a două interogări într-un singur set de rezultate
32. Sintaxa corectă pentru eliminarea valorilor nule din rezultatele interogării este
    a. = NULL
    b. IS NULL
    c. NOT = NULL
    d. IS NOT NULL
    e. <> NULL

33. O instrucțiune SQL care conține o funcție de agregare
    a. Trebuie să includă o clauză GROUP BY
    b. Nu poate include, în același timp, o clauză GROUP BY și o clauză ORDER BY
    c. Trebuie să includă o clauză ORDER BY
    d. Poate conține și coloane calculate
    e. Poate conține și coloane obișnuite
34. O subinterogare
    a. Trebuie să fie încadrată în paranteze
    b. Reprezintă o cale puternică de calculare a coloanelor
    c. Poate fi corelată sau necorelată
    d. Permite selectarea flexibilă a rândurilor
    e. Poate fi folosită pentru a selecta valorile care vor fi aplicate condițiilor din clauza WHERE
35. O interogare fără o clauză WHERE sau o clauză JOIN
    a. Realizează o uniune externă (outer join)
    b. Realizează o uniune internă (inner join)
    c. Are ca rezultat un mesaj de eroare
    d. Are ca rezultat un produs cartezian
    e. Nu returnează nici un rând din setul de rezultate
36. O auto-uniune (self join)
    a. Poate fi o uniune internă sau o uniune externă
    b. Rezolvă relațiile recursive
    c. Nu poate avea niciodată ca rezultat un produs cartezian
    d. Implică două tabele diferite
    e. Poate folosi o subinterogare pentru a limita și mai mult rândurile selectate
37. O uniune (join)
    a. Necesită folosirea unei clauze JOIN
    b. Necesită o listă de tabele separate prin virgule în clauza FROM
    c. Combină rânduri din interogări multiple într-un singur set de rezultate
    d. Combină coloanele din două sau mai multe tabele în rezultatele unei singure interogări
    e. Este realizată ori de câte ori în clauza FROM sunt specificate mai multe tabele

38. Calificatorii numelor de coloane
    a. Pot fi pseudonime pentru numele de coloane, definite în clauza FROM
    b. Pot fi pseudonime pentru numele de tabele, definite în clauza FROM
    c. Pot fi nume de tabele
    d. Rezolvă referințele ambigue la coloane
    e. Pot fi numere care indică poziția relativă a tabelelor în lista FROM
39. O subinterogare corelată
    a. Are o interogare internă care referă coloane din interogarea externă
    b. Are o interogare internă care nu face nici o referire la coloanele din interogarea externă
    c. Rulează mai eficient decât o subinterogare necorelată
    d. Rulează mai puțin eficient decât o subinterogare necorelată
    e. Are o interogare externă care referă coloane din interogarea internă
40. Funcțiile SQL
    a. Pot fi folosite în clauza WHERE a unei instrucțiuni SQL
    b. Pot fi folosite ca pseudonime pentru numele tabelelor într-o instrucțiune SQL
    c. Pot fi folosite în lista de coloane a unei instrucțiuni SQL
    d. Returnează un set de valori
    e. Returnează o singură valoare
41. Funcția RTRIM
    a. Elimină spațiile de la sfârșitul șirurilor de caractere
    b. Elimină spațiile de la începutul șirurilor de caractere
    c. Poate fi imbricată cu alte funcții
    d. Înlocuiește valorile nule cu alte valori în șirurile de caractere
    e. Elimină spațiile de la începutul și de la sfârșitul șirurilor de caractere
42. Expresiile CASE
    a. Există sub două forme, respectiv statice și dinamice
    b. Există sub două forme, respectiv simple și cu căutare
    c. Există sub două forme, respectiv standard și cu căutare
    d. Există sub două forme, respectiv cu căutare și fără căutare
    e. Permit executarea condițională a clauzelor dintr-o instrucțiune SQL

43. O instrucțiune DML poate referi
    a. Coloane ale unei vizualizări bazate pe mai multe tabele
    b. Coloane ale unei vizualizări bazate pe un singur tabel
    c. Coloane din mai multe tabele
    d. Coloane dintr-un singur tabel
    e. O vizualizare care conține coloane dintr-un singur tabel
44. La formarea unei instrucțiuni DML, trebuie să țineți seama de următoarele restricții
    a. Restricții referențiale
    b. Restricții cheie primară
    c. Restricții NOT NULL
    d. Restricții de unicitate
    e. Restricții de securitate
45. O instrucțiune INSERT cu o clauză VALUES
    a. Poate folosi cuvântul cheie NULL pentru a atribui valori nule coloanelor
    b. Poate insera rânduri multiple prin rularea unei singure instrucțiuni
    c. Trebuie să aibă o listă de coloane
    d. Trebuie să aibă o listă de valori
    e. Poate include o clauză WHERE
46. O instrucțiune UPDATE fără o clauză WHERE
    a. Eșuează și returnează o eroare
    b. Are ca rezultat un produs cartezian
    c. Actualizează toate rândurile din tabel cu valori nule
    d. Încearcă să actualizeze toate rândurile din tabel
    e. Șterge toate rândurile din tabel
47. O instrucțiune UPDATE
    a. Trebuie să includă o clauză WHERE
    b. Poate atribui unei coloane valoarea unei alte coloane
    c. Poate atribui valori noi coloanelor din mai multe tabele
    d. Trebuie să includă o clauză SET
    e. Trebuie să furnizeze o nouă valoare pentru cel puțin o coloană

48. O instrucțiune DELETE
    a. Poate include o clauză WHERE opțională
    b. Poate folosi cuvântul cheie FORCE pentru a forța ștergerea rândurilor
    c. Poate include o listă opțională de coloane
    d. Nu poate încălca restricțiile referențiale ale tabelului
    e. Poate avea o instrucțiune SELECT internă, ca parte a clauzei WHERE
49. Clauza SET dintr-o instrucțiune UPDATE poate atribui unei coloane o valoare care este
    a. O listă de valori
    b. O constantă
    c. Numele unei alte coloane
    d. Orice expresie din care rezultă o singură valoare
    e. Cuvântul cheie NULL
50. Securitatea este necesară pentru că
    a. Persoanele oneste fac greșeli
    b. Bazele de date conectate la Internet sunt vulnerabile
    c. 80% dintre fraude sunt comise de hackeri din exterior
    d. Controalele de securitate ale aplicațiilor nu sunt adecvate
    e. Controalele de securitate mențin onestitatea persoanelor
51. Între intrușii care încearcă să penetreze sistemele conectate la Internet se numără
    a. Angajați nemulțumiți
    b. Spioni ai concurenței
    c. Experți în audit bancar
    d. Autori de jurnale personale web
    e. Hackeri
52. Componentele care trebuie protejate sunt
    a. Rețelele
    b. Sistemele de operare
    c. Serverele
    d. Stațiile de lucru client
    e. Bazele de date

53. Privilegiile de sistem
    a. Sunt specifice unui obiect al bazei de date
    b. Variază între bazele de date provenite de la furnizori diferiți
    c. Sunt acordate într-o manieră similară în Oracle, Sybase și Microsoft SQL Server
    d. Sunt anulate cu ajutorul instrucțiunii SQL REMOVE
    e. Permit beneficiarului să efectueze o serie de funcții administrative pe server, cum ar fi oprirea serverului
54. Privilegiile de obiecte
    a. Sunt specifice unui obiect al bazei de date
    b. Permit beneficiarului să efectueze o serie de funcții administrative pe server, cum ar fi oprirea serverului
    c. Sunt acordate într-o manieră similară în Oracle, Sybase și Microsoft SQL Server
    d. Sunt atribuite cu ajutorul instrucțiunii SQL GRANT
    e. Sunt anulate cu ajutorul instrucțiunii SQL REMOVE
55. Utilizarea clauzei WITH GRANT OPTION la atribuirea privilegiilor de obiecte
    a. Este o procedură recomandată pe scară largă pentru că este comodă în utilizare
    b. Va fi revocată în cascadă dacă privilegiul este ulterior revocat
    c. Acordă privilegiile DBA ale beneficiarului întregii baze de date
    d. Permite beneficiarului să acorde privilegiul altor utilizatori
    e. Poate conduce la probleme de securitate
56. Rolurile
    a. Pot exista înainte de existența utilizatorilor
    b. Pot conține oricâte privilegii de obiecte
    c. Pot conține un singur privilegiu de obiecte
    d. Pot fi atribuite unui singur utilizator
    e. Pot fi partajate de mai mulți utilizatori
57. Posibilele dezavantaje ale utilizării rolurilor pentru securitate sunt
    a. Sunt eliminate atunci când este șters utilizatorul
    b. Sunt eliminate atunci când sunt șterse privilegiile
    c. Pot fi atribuite fără a lua în considerare toate privilegiile conținute

d. Sunt mai dificil de administrat decât privilegiile individuale
e. Este necesară o perioadă suplimentară de instruire pentru administratorii care trebuie să le folosească

58. Vizualizările pot ajuta la implementarea politicii de securitate prin
a. Stocarea rezultatelor auditării bazei de date
b. Restricționarea rândurilor dintr-un tabel la care are acces un utilizator
c. Restricționarea coloanelor dintr-un tabel la care are acces un utilizator
d. Restricționarea bazelor de date la care are acces un utilizator
e. Monitorizarea intrușilor la baza de date

59. O tranzacție
a. Schimbă baza de date dintr-o stare consistentă în alta
b. Are proprietățile descrise de acronimul ACID
c. Nu poate fi procesată parțial
d. Poate fi procesată parțial
e. Este denumită uneori *unitate de lucru*

60. Procesul care anulează modificările efectuate de o tranzacție eșuată este numit
a. Recuperare
b. Crearea unui punct de salvare
c. Înregistrarea tranzacției
d. Finalizare
e. Refacere

61. Procesul care face permanente modificările efectuate de o tranzacție este numit
a. Recuperare
b. Crearea unui punct de salvare
c. Înregistrarea tranzacției
d. Finalizare
e. Refacere

62. Suportul pentru tranzacții în bazele de date relaționale include:
a. Managementul bazei de date distribuite
b. Înregistrarea fiecărei tranzacții în jurnalul de tranzacții
c. Identificarea începutului fiecărei tranzacții
d. Identificarea sfârșitului fiecărei tranzacții
e. Salvările periodice ale bazei de date

63. Cantitatea de date afectată de o blocare (granularitatea blocării) poate fi
    a. O coloană
    b. Un rând
    c. Un tabel
    d. Un bloc sau o pagină
    e. O bază de date
64. Problema actualizării concurente
    a. Nu poate apărea când este activată (ON) opțiunea AUTOCOMMIT
    b. Apare când doi utilizatori ai bazei de date inițiază instrucțiuni SELECT care intră în conflict
    c. Apare când doi utilizatori ai bazei de date efectuează actualizări conflictuale aceloraşi date
    d. Este o consecință a partajării simultane a datelor
    e. Este motivul pentru care trebuie asigurat suport pentru blocarea tranzacțiilor
65. O blocare
    a. Poate provoca apariția unor conflicte atunci când alți utilizatori încearcă să actualizeze datele blocate
    b. Poate avea niveluri și un protocol de extindere în unele produse RDBMS
    c. Are setat un interval de așteptare în sistemul DB2 și în alte produse RDBMS
    d. Este un element de control atașat datelor pentru a le rezerva în scopul actualizării de către utilizator
    e. Este anulată de obicei atunci când are loc o operație COMMIT sau ROLLBACK
66. O interblocare
    a. Poate, teoretic, să plaseze doi sau mai mulți utilizatori într-o stare de așteptare infinită
    b. Este o blocare ce a depășit intervalul de așteptare și din acest motiv nu mai este necesară
    c. Poate fi rezolvată prin detectarea interblocărilor în unele sisteme RDBMS
    d. Poate fi rezolvată prin intervale de așteptare a blocării în unele sisteme RDBMS
    e. Apare când doi utilizatori ai bazei de date solicită simultan blocarea unor date blocate de celălalt utilizator

67. Un cursor este
    a. Un pointer (indicator) într-un set de rezultate
    b. Același lucru cu un set de rezultate
    c. O metodă de analiză a performanței instrucțiunilor SQL
    d. Colecția de rânduri returnate de o interogare la baza de date
    e. Un buffer care păstrează rândurile extrase din baza de date
68. Un set de rezultate este
    a. Un pointer într-un cursor
    b. O metodă de analiză a performanței instrucțiunilor SQL
    c. Același lucru cu un cursor
    d. Un buffer care păstrează rândurile extrase din baza de date
    e. Colecția de rânduri returnate de o interogare la baza de date
69. Înainte ca rândurile să poată fi extrase dintr-un cursor, cursorul trebuie să fie
    a. Deschis
    b. Închis
    c. Dezalocat
    d. Declarat
    e. Finalizat
70. Cursoarele sunt
    a. Incluse în limbajul Oracle PL/SQL
    b. Incluse în limbajul Sybase Transact-SQL
    c. Incluse în limbajul Microsoft Transact-SQL
    d. Destinate să anuleze nepotrivirea dintre modul în care tratează rezultatele unei interogări limbajele orientate spre obiecte și bazele de date relaționale
    e. Destinate să anuleze nepotrivirea dintre modul în care tratează rezultatele unei interogări limbajele procedurale și bazele de date relaționale
71. Numele cursorului este inclus în
    a. Instrucțiunea OPEN
    b. Instrucțiunea FETCH
    c. Instrucțiunea CLOSE
    d. Instrucțiunea DECLARE CURSOR
    e. Instrucțiunea SELECT

72. O instrucțiune OPEN pentru cursor
    a. Trebuie să includă o clauză INTO
    b. Trebuie să includă numele cursorului
    c. Trebuie să includă numele tabelului (tabelelor)
    d. Poate determina rularea interogării și plasarea unor date în setul de rezultate
    e. Determină întotdeauna rularea interogării și completarea setului de rezultate cu date
73. ODBC este
    a. Un standard Microsoft
    b. Independent de orice limbaj, sistem de operare sau sistem DBMS
    c. O interfață API standard pentru conectarea la sistemele DBMS
    d. Flexibil în tratarea limbajelor SQL proprietare
    e. Utilizat de programele Java
74. JDBC este
    a. Un standard Microsoft
    b. Independent de orice limbaj, sistem de operare sau sistem DBMS
    c. O interfață API standard pentru conectarea la sistemele DBMS
    d. Flexibil în tratarea limbajelor SQL proprietare
    e. Utilizat de programele Java
75. O procedură de declanșare este
    a. Scrisă într-un limbaj neprocedural
    b. Scrisă într-un limbaj procedural
    c. Stocată în baza de date
    d. Executată doar atunci când este apelată
    e. Executată automat pe baza unui eveniment din baza de date
76. O procedură memorată este
    a. Scrisă într-un limbaj neprocedural
    b. Scrisă într-un limbaj procedural
    c. Stocată în baza de date
    d. Executată doar atunci când este apelată
    e. Executată automat pe baza unui eveniment din baza de date

77. Cerințele pentru performanță
    a. Asigură o cale de a afla când să se încheie particularizarea unei interogări
    b. Sunt mai bune atunci când conțin criterii complexe
    c. Ar trebui stabilite după ajustarea instrucțiunilor SQL
    d. Asigură o cale de identificare a instrucțiunilor ce necesită ajustare
    e. Sunt dezvoltate doar pentru a-i face pe auditori fericiți
78. Citirile și scrierile efectuate pe disc pot fi diminuate
    a. Plasând toate fișierele bazei de date pe o singură unitate de disc
    b. Introducând toată baza de date în memorie
    c. Dispersând fișierele pe toate unitățile de disc disponibile
    d. Alocând buffere de capacitate suficientă
    e. Adăugând indexuri pentru toate coloanele importante din tabel
79. Un proiect eficient de tabel include
    a. Folosirea de tipuri de date identice pentru cheile primare și pentru cheile externe corespunzătoare
    b. Folosirea de tipuri de date identice pentru toate coloanele cu cheie primară
    c. Folosirea de proceduri de declanșare oricând este posibil
    d. Folosirea tipului VARCHAR pentru toate de tip caracter cu lungime variabilă
    e. Folosirea celui mai mic tip de dată numerică posibil care reține valorile datelor
80. Optimizatorul de interogări
    a. Poate folosi date statistice adunate din baza de date
    b. Poate folosi reguli aplicate modului în care a fost scrisă instrucțiunea
    c. Poate fi bazat pe cost sau bazat pe reguli
    d. Creează un plan de execuție a interogării în tabelul de planuri
    e. Determină cea mai bună cale de execuție a unei instrucțiuni SQL
81. Scanările unui tabel pot fi evitate
    a. Asigurându-vă că cel puțin un predicat face referire la coloana principală a unui index
    b. Evitând coloanele care nu sunt folositoare
    c. Folosind GROUP BY în loc de DISTINCT

d. Incluzând o clauză WHERE care referă o coloană indexată
e. Asigurându-vă că datele statistice sunt actualizate

82. Un index nu poate fi folosit când
    a. Operatorul NOT EQUAL este folosit într-un predicat
    b. Operatorul NOT este folosit într-un predicat
    c. O funcție SQL este inclusă într-o comparație de coloane (exceptând cazul în care corespunde unui index bazat pe funcții)
    d. Clauza WHERE face referire la a doua coloană a unui index
    e. Există o clauză LIKE care face referire la un șir de comparație care conține un caracter de înlocuire (cu excepția primei poziții a șirului)
83. Aspectele privind folosirea indexurilor includ
    a. Evitarea suprapunerii indexurilor
    b. Evitarea indexurilor unice
    c. Amplasarea de indexuri pentru toate coloanele actualizate frecvent
    d. Amplasarea de indexuri pentru coloanele cu cheie externă
    e. Crearea de indexuri pentru coloanele care au doar câteva valori posibile

Scrieți instrucțiunea SQL pentru fiecare dintre următoarele probleme.

84. Găsiți toate filmele din tabelul MOVIE pentru care valoarea MPAA_RATING_CODE este G, P sau PG-13. Nu folosiți operatorul OR.
85. Câte filme din tabelul MOVIE au cuvântul „of" în titlu? Asigurați-vă că tratați atât cazurile în care cuvântul „of" este primul sau ultimul cuvânt al titlului, cât și cazurile în care se găsește în interiorul titlului și cele în care apare cu majuscule.
86. Câte conturi de client sunt închise conform valorii datei din coloana DATE_TERMINATED?
87. Găsiți valorile minimă și maximă pentru prețul VHS (RETAIL_PRICE_VHS) din tabelul MOVIE.
88. Găsiți prețul DVD mediu (RETAIL_PRICE_DVD) pentru fiecare gen de film (MOVIE_GENRE_CODE) din tabelul MOVIE. Rotunjiți prețul mediu cu două zecimale exacte.
89. Folosiți funcția SUM pentru a afla totalul valorilor din coloana CUSTOMER_DEPOSIT_AMOUNT din tabelul CUSTOMER_ACCOUNT.

90. Folosiți o subinterogare pentru a lista valorile Movie ID și Title corespunzătoare tuturor filmelor închiriate.
91. Folosiți o uniune cu predicatul de uniune în clauza WHERE pentru a lista valorile Movie ID și Title corespunzătoare tuturor filmelor închiriate. Verificați ca filmele închiriate de mai multe ori să fie afișate o singură dată în setul de rezultate.
92. Listați titlul pentru fiecare film împreună cu totalul copiilor în format DVD din inventar.
93. Listați toate identificatoarele de conturi client împreună cu numărul de împrumuturi efectuate de fiecare cont în parte.
94. Aflați prețul mediu al unui film pe format DVD (coloana RETAIL_PRICE_DVD din tabelul MOVIE), cu media calculată rotunjită prin scădere la cea mai apropiată sumă fără zecimale.
95. Aflați valoarea ASCII a caracterului punct și virgulă (;).
96. Listați valorile EMPLOYEE_TAX_ID din tabelul EMPLOYEES, cu cratimele transformate în spații.
97. Scrieți o instrucțiune SQL care listează titlurile tuturor filmelor, cu prețul formatului DVD (RETAIL_PRICE_DVD) organizat astfel: Nul (valoare nulă sau 0), Profitabil (valoarea < 20), Mediu (valoarea >= 20 și < 30) și Excelent (valoarea >= 30).
98. Folosind o instrucțiune INSERT cu o clauză VALUES și o listă de coloane, introduceți un rând nou în tabelul MOVIE, având următoarele valori pentru date:

MOVIE_ID	999
MOVIE_GENRE_CODE	Drama
MPAA_RATING_CODE	NR
MOVIE_TITLE	How the Database Was Won

99. Actualizați tabelul MOVIE pentru a mări cu valoarea 1 prețurile diferite de 0 ale tuturor filmelor pe format DVD (RETAIL_PRICE_DVD).
100. Ștergeți toate rândurile din tabelul MOVIE_GENRE care nu sunt referite de un rând din tabelul MOVIE.

Răspunsurile la chestionare și la examenul final

Capitolul 1

1. b, d
2. c, e
3. a, b
4. c, d
5. a, b, c, d, e
6. a, c, d
7. a, b, e
8. c, d
9. a, c, d, e
10. b, c, d
11. a, d, e
12. b, d
13. a, c, e
14. b, d
15. a
16. c
17. a, b, c, d, e
18. b, d, e
19. a, c, d
20. b, e

Capitolul 2

1. a, b, c, e
2. c, d, e
3. b, d
4. a, c, d
5. b, e
6. a, d
7. b, c, d
8. c, d, e
9. a, b

10. a, d
11. b, c, d, e
12. b, c, d
13. a, d
14. a, b, c, e
15. c
16. b, c, d
17. a, d, e
18. c, d
19. a
20. b, e

Capitolul 3

1. b, d, e
2. a, b, e
3. d, e
4. a, c
5. b, c, e
6. a, d
7. a, b
8. b, d
9. b, d, e
10. a, c
11. b, c
12. c
13. b
14. a
15. d
16. b, e
17. a, c, e
18. a, b, d, e
19. a, c, e
20. b, c, e

Capitolul 4

1. c
2. b, d
3. a, c
4. b, e
5. a, b
6. b, e
7. e
8. a, c, d
9. b, e
10. a, d
11.

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE <> 'R';
```

12.

```
SELECT MOVIE_TITLE, RETAIL_PRICE_DVD
  FROM MOVIE
 WHERE RETAIL_PRICE_DVD BETWEEN 19.99 AND 29.99;
```

13.

```
SELECT MOVIE_TITLE, MOVIE_GENRE_CODE, MPAA_RATING_CODE
  FROM MOVIE
 WHERE (MOVIE_GENRE_CODE = 'Comdy' AND
        MPAA_RATING_CODE = 'PG-13')
    OR (MOVIE_GENRE_CODE = 'Drama' AND
        MPAA_RATING_CODE = 'R');
```

14.

```
SELECT COUNT(*)
  FROM MOVIE_RENTAL
 WHERE LATE_OR_LOSS_FEE IS NULL;
```

15.

```
SELECT COUNT(*)
  FROM PERSON
 WHERE UPPER(PERSON_FAMILY_NAME) LIKE '%A%';
```

16.

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE UPPER(MOVIE_TITLE) LIKE '% THE %'
    OR UPPER(MOVIE_TITLE) LIKE 'THE %'
    OR UPPER(MOVIE_TITLE) LIKE '% THE';
```

17.

```
SELECT SUM(RENTAL_FEE) AS TOTAL_RENTAL_FEES
  FROM MOVIE_RENTAL;
```

18. Pentru Oracle și DB2:

```
SELECT DISTINCT SUBSTR(PERSON_FAMILY_NAME,1,5)
  FROM PERSON;
```

Pentru Microsoft SQL Server, Sybase Adaptive Server și MySQL:

```
SELECT DISTINCT SUBSTRING(PERSON_FAMILY_NAME,1,5)
  FROM PERSON;
```

19.

```
SELECT MOVIE_GENRE_CODE,
       ROUND(AVG(RETAIL_PRICE_DVD),2) AS AVG_PRICE
  FROM MOVIE
 GROUP BY MOVIE_GENRE_CODE
```

20. Pentru Oracle:

```
SELECT MOVIE_ID,
       SUM(RENTAL_FEE + NVL(LATE_OR_LOSS_FEE,0))
           AS TOTAL_FEES
  FROM MOVIE_RENTAL
 GROUP BY MOVIE_ID;
```

Pentru Microsoft SQL Server și Sybase Adaptive Server:

```
SELECT MOVIE_ID,
       SUM(RENTAL_FEE + ISNULL(LATE_OR_LOSS_FEE,0))
           AS TOTAL_FEES
  FROM MOVIE_RENTAL
 GROUP BY MOVIE_ID;
```

Pentru MySQL:

```
SELECT MOVIE_ID,
       SUM(RENTAL_FEE + IFNULL(LATE_OR_LOSS_FEE,0))
           AS TOTAL_FEES
  FROM MOVIE_RENTAL
 GROUP BY MOVIE_ID;
```

Capitolul 5

1. a, e
2. e
3. b, d, e
4. b, c, d
5. a, c
6. b, d
7. a, b, e
8. b
9. a, c
10. b, c
11.

```
SELECT PERSON_GIVEN_NAME, PERSON_FAMILY_NAME,
       CUSTOMER_ACCOUNT_ID
  FROM PERSON JOIN CUSTOMER_ACCOUNT_PERSON
       USING (PERSON_ID)
 WHERE DEATH_DATE IS NOT NULL;
```

sau:

```
SELECT PERSON_GIVEN_NAME, PERSON_FAMILY_NAME,
       CUSTOMER_ACCOUNT_ID
  FROM PERSON A JOIN CUSTOMER_ACCOUNT_PERSON B
       ON A.PERSON_ID = B.PERSON_ID
 WHERE DEATH_DATE IS NOT NULL;
```

12.

```
SELECT PERSON_GIVEN_NAME, PERSON_FAMILY_NAME
  FROM PERSON
    JOIN EMPLOYEE USING (PERSON_ID)
    JOIN CUSTOMER_ACCOUNT_PERSON USING (PERSON_ID);
```

sau:

```
SELECT PERSON_GIVEN_NAME, PERSON_FAMILY_NAME
  FROM PERSON A
    JOIN EMPLOYEE B ON A.PERSON_ID = B.PERSON_ID
    JOIN CUSTOMER_ACCOUNT_PERSON C
                    ON A.PERSON_ID = C.PERSON_ID;
```

13.

```
SELECT PERSON_GIVEN_NAME, PERSON_FAMILY_NAME
  FROM PERSON
 WHERE PERSON_ID IN
       (SELECT PERSON_ID FROM EMPLOYEE)
   AND PERSON_ID IN
       (SELECT DISTINCT PERSON_ID
          FROM CUSTOMER_ACCOUNT_PERSON);
```

14.

```
SELECT C.PERSON_FAMILY_NAME AS EMPLOYEE_NAME,
       D.PERSON_FAMILY_NAME AS SUPERVISOR_NAME
  FROM EMPLOYEE A
         JOIN EMPLOYEE B ON (A.SUPERVISOR_PERSON_ID =
                             B.PERSON_ID)
         JOIN PERSON C ON (A.PERSON_ID = C.PERSON_ID)
         JOIN PERSON D ON (B.PERSON_ID = D.PERSON_ID);
```

15.

```
SELECT MEDIA_FORMAT, COUNT(*)
  FROM MOVIE_COPY
    JOIN MOVIE_RENTAL USING (MOVIE_ID, COPY_NUMBER)
 GROUP BY MEDIA_FORMAT;
```

sau:

```
SELECT MEDIA_FORMAT, COUNT(*)
  FROM MOVIE_COPY A JOIN MOVIE_RENTAL B
       ON A.MOVIE_ID = B.MOVIE_ID AND
          A.COPY_NUMBER = B.COPY_NUMBER
 GROUP BY MEDIA_FORMAT;
```

16.

```
SELECT MOVIE_ID, DUE_DATE
  FROM MOVIE_RENTAL
    JOIN MOVIE_COPY USING (MOVIE_ID, COPY_NUMBER)
 WHERE RETURNED_DATE IS NULL
   AND DATE_SOLD IS NULL;
```

sau:

```
SELECT A.MOVIE_ID, DUE_DATE
  FROM MOVIE_RENTAL A JOIN MOVIE_COPY B
       ON A.MOVIE_ID = B.MOVIE_ID AND
          A.COPY_NUMBER = B.COPY_NUMBER
 WHERE RETURNED_DATE IS NULL
   AND DATE_SOLD IS NULL;
```

17.

```
SELECT A.MOVIE_ID, A.MOVIE_TITLE
  FROM MOVIE A
 WHERE MOVIE_ID NOT IN
       (SELECT DISTINCT MOVIE_ID
          FROM MOVIE_RENTAL);
```

18.

```
SELECT MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_ID IN
       (SELECT MOVIE_ID
          FROM MOVIE_COPY
         WHERE MEDIA_FORMAT = 'V'
           AND DATE_SOLD IS NULL)
   AND MOVIE_ID NOT IN
       (SELECT MOVIE_ID
          FROM MOVIE_COPY
         WHERE MEDIA_FORMAT = 'D'
           AND DATE_SOLD IS NULL);
```

19.

```
SELECT MOVIE_TITLE
  FROM MOVIE A
    JOIN MOVIE_COPY B ON
         (A.MOVIE_ID = B.MOVIE_ID AND
         MEDIA_FORMAT = 'V' AND
         DATE_SOLD IS NULL)
 WHERE A.MOVIE_ID NOT IN
     (SELECT MOVIE_ID
        FROM MOVIE_COPY
       WHERE MEDIA_FORMAT = 'D'
         AND DATE_SOLD IS NULL)
```

20.

```
SELECT MOVIE_ID, MEDIA_FORMAT, COUNT(*)
  FROM MOVIE_COPY
 GROUP BY MOVIE_ID, MEDIA_FORMAT
HAVING COUNT(*) >= 2;
```

Capitolul 6

1. b, c, e
2. c
3. a, c

4. b, c
5. a, d
6. a, b
7. b, d, e
8. a, d
9. b, c, d, e
10. a, d
11.

```
SELECT REPLACE(MPAA_RATING_CODE,'-',' ')
       AS MPAA_RATING_CODE
  FROM MPAA_RATING;
```

12. Oracle:

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%' || CHR(39) || '%';
```

Microsoft SQL Server:

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%' + CHAR(39) + '%';
```

Alte implementări SQL:

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE
 WHERE MOVIE_TITLE LIKE '%' || CHAR(39) || '%'
```

NOTĂ: Această instrucțiune pare să nu lucreze corect în MySQL 4.

13.

```
SELECT ASCII('!');
```

NOTĂ: În sintaxa Oracle este necesară o clauză FROM în toate instrucțiunile SELECT, astfel că trebuie adăugată clauza „FROM DUAL" pentru ca această instrucțiune să lucreze în Oracle.

14. Oracle:

```
SELECT CEIL(AVG(RETAIL_PRICE_DVD)) AS AVERAGE_DVD_PRICE
  FROM MOVIE;
```

Majoritatea celorlalte implementări SQL:

```
SELECT CEILING(AVG(RETAIL_PRICE_DVD)) AS AVERAGE_DVD_PRICE
  FROM MOVIE;
```

15.

```
SELECT FLOOR(AVG(RETAIL_PRICE_VHS)) AS AVERAGE_VHS_PRICE
  FROM MOVIE;
```

16.

```
SELECT 'ALTER TABLE ' || TABLE_NAME ||
       ' DROP CONSTRAINT ' || CONSTRAINT_NAME || ';'
  FROM USER_CONSTRAINTS
 WHERE CONSTRAINT_TYPE = 'R';
```

17.

```
SELECT NAME FROM SYSOBJECTS
 WHERE XTYPE = 'F';
```

18.

```
SELECT CUSTOMER_ACCOUNT_ID,
  CASE CHILD_RENTAL_ALLOWED_INDIC
    WHEN 'Y' THEN 'Child Rental OK'
    WHEN 'N' THEN 'NO CHILD RENTAL'
    END AS CHILD_RENTAL
  FROM CUSTOMER_ACCOUNT
 ORDER BY CUSTOMER_ACCOUNT_ID;
```

19.

```
SELECT MOVIE_ID, YEAR_PRODUCED,
   CASE
      WHEN YEAR_PRODUCED IS NULL THEN 'Unknown'
      WHEN YEAR_PRODUCED BETWEEN '1980' and '1989'
           THEN '80s'
      WHEN YEAR_PRODUCED BETWEEN '1990' and '1999'
           THEN '80s'
      WHEN YEAR_PRODUCED BETWEEN '2000' and '2009'
           THEN '00s'
   END AS DECADE
  FROM MOVIE
 ORDER BY MOVIE_ID;
```

20.

```
SELECT MOVIE_ID, COPY_NUMBER, TRANSACTION_ID,
   CASE
      WHEN LATE_OR_LOSS_FEE IS NULL THEN 'None'
      WHEN LATE_OR_LOSS_FEE = 0 THEN 'None'
      WHEN LATE_OR_LOSS_FEE < 10 THEN 'Minor'
```

```
      WHEN LATE_OR_LOSS_FEE >= 10 THEN 'Major'
   END AS FEE_CATEGORY
  FROM MOVIE_RENTAL
 ORDER BY MOVIE_ID, COPY_NUMBER, TRANSACTION_ID;
```

Capitolul 7

1. a, c
2. b, c, e
3. a, c, d, e
4. b, d
5. a, c, e
6. c, d, e
7. b
8. c
9. a, b, d
10. b, d, e
11. a, b, d, e
12. a, b, c, d, e
13.

```
INSERT INTO MOVIE_GENRE
VALUES ('TRAIN','Training');
```

14.

```
INSERT INTO MOVIE
(MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE)
VALUES (99, 'TRAIN', 'NR', 'Employee Training Video');
```

15.

```
INSERT INTO MOVIE_LANGUAGE
SELECT MOVIE_ID, 'ja'
  FROM MOVIE;
```

16.

```
INSERT INTO RENTAL_TOTAL
SELECT MOVIE_ID, COUNT(*), SUM(RENTAL_FEE)
  FROM MOVIE_RENTAL
 GROUP BY MOVIE_ID;
```

17.

```
DELETE FROM RENTAL_TOTAL;
```

18.

```
DELETE FROM MOVIE_LANGUAGE
 WHERE LANGUAGE_CODE = 'ja';
```

19.

```
UPDATE MOVIE_COPY
   SET DATE_SOLD = '2005-01-15'
 WHERE MOVIE_ID = 1
   AND COPY_NUMBER = 1;
```

20.

```
UPDATE MOVIE
   SET RETAIL_PRICE_VHS = ROUND(RETAIL_PRICE_VHS * 1.1, 2)
 WHERE RETAIL_PRICE_VHS IS NOT NULL;
```

Capitolul 8

1. b, e
2. a, b, d, e
3. b, d, e
4. a, b, c, d, e
5. a, c, d
6. b, c, e
7. b, c, d, e
8. b, c, e
9. a, c, e
10. a, b, e
11. a, c, d
12. b, c, d
13. d, e
14. a, c
15. c
16. d
17. c
18. e
19.

```
GRANT SELECT, INSERT, DELETE
   ON EMPLOYEE
   TO manager_1, manager_2;
```

20.

```
REVOKE INSERT, UPDATE, DELETE
    ON MOVIE
  FROM clerk_127;
```

Capitolul 9

1. b, c, d, e
2. c
3. b
4. d
5. a
6. c
7. b
8. a, b, e
9. a, d, e
10. a, e
11. d
12. a, b, c
13. a, c, d
14. c, d, e
15. b, d
16. c, d, e
17. a, b, c, d, e
18. a, c, e
19. b, c, d, e
20. b, d, e

Capitolul 10

1. b
2. a
3. a, c
4. b, c, d, e
5. a, c, d, e
6. e

7. b, d
8. a, b, e
9. a, d, e
10. b, c
11. a, b, e
12. b, d, e
13. b, c, d, e
14. b, d, e
15. a, d, e
16. c, e
17. a, b, c, d
18. b, d, e
19. a, c, d
20. b, d, e

Capitolul 11

1. b, d
2. a, c, e
3. c, e
4. b, d
5. a, c, d
6. b, c, d, e
7. a, b, d
8. a, c, d
9. a, c, d, e
10. b, c
11. b, d
12. a, e
13. c
14. a, c, d
15. b, d
16. a, d
17. a, b

18. a, c
19. c, d
20. b

Răspunsurile la examenul final

1. c, d, e
2. b, e
3. c
4. a, c
5. a, c, e
6. b, c, d, e
7. b, e
8. a, b, c, d
9. c, d, e
10. a, b, c, d, e
11. a, b, d, e
12. a, b, e
13. b, d
14. a, b, d, e
15. a, b, c
16. a, b
17. c
18. d, e
19. a, d
20. d, e
21. a, c, e
22. b, d
23. a, b, e
24. b, c
25. c, d
26. d
27. b
28. c

29. a
30. b, d, e
31. c, e
32. d
33. d, e
34. a, d, e
35. b ,d
36. a, b, e
37. d, e
38. b, c, d
39. d, e
40. a, c, e
41. a, c
42. b, e
43. b, d, e
44. a, b, c, d
45. a, d
46. d
47. b, d, e
48. a, d, e
49. b, c, d, e
50. a, b, c, d, e
51. a, b, e
52. a, b, c, d, e
53. b, c, e
54. a, c, d
55. b, d, e
56. a, b, e
57. c, e
58. b, c
59. a, b, c, e
60. e
61. d
62. b, c, d

63. a, b, c, d, e
64. c, d, e
65. a, b, c, d, e
66. a, c, d, e
67. a
68. e
69. a, d
70. a, b, c, e
71. a, b, c, d
72. b, d
73. b, c, d
74. c, d, e
75. b, c, e
76. b, c, d
77. a, d
78. b, c, d
79. a, e
80. a, b, c, e
81. a, d, e
82. a, b, c, d
83. a, d,
84.

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE
 WHERE MPAA_RATING_CODE IN ('G','PG','PG-13');
```

85.

```
SELECT COUNT(*)
  FROM MOVIE
 WHERE LOWER(MOVIE_TITLE) LIKE 'of %'
    OR LOWER(MOVIE_TITLE) LIKE '% of'
    OR LOWER(MOVIE_TITLE) LIKE '% of %';
```

86.

```
SELECT COUNT(*)
  FROM CUSTOMER_ACCOUNT
 WHERE DATE_TERMINATED IS NOT NULL;
```

87.

```
SELECT MIN(RETAIL_PRICE_VHS) AS MIN_PRICE,
   MAX(RETAIL_PRICE_VHS) AS MAX_PRICE
  FROM MOVIE;
```

88.

```
SELECT MOVIE_GENRE_CODE,
       ROUND(AVG(RETAIL_PRICE_DVD),2) AS AVG_PRICE
  FROM MOVIE
 GROUP BY MOVIE_GENRE_CODE;
```

89.

```
SELECT SUM(CUSTOMER_DEPOSIT_AMOUNT) AS TOTAL
  FROM CUSTOMER_ACCOUNT;
```

90.

```
SELECT MOVIE_ID, MOVIE_TITLE
  FROM MOVIE A
 WHERE EXISTS
   (SELECT MOVIE_ID
  FROM MOVIE_RENTAL B
 WHERE A.MOVIE_ID = B.MOVIE_ID);
```

91.

```
SELECT A.MOVIE_ID, A.MOVIE_TITLE
  FROM MOVIE A, MOVIE_RENTAL B
 WHERE A.MOVIE_ID = B.MOVIE_ID
 GROUP BY A.MOVIE_ID, A.MOVIE_TITLE;
```

Cuvântul cheie DISTINCT poate fi folosit în locul clauzei GROUP BY, însă, de obicei este mai puțin eficient.

92.

```
SELECT MOVIE_TITLE, COUNT(*) AS DVD_COPIES
  FROM MOVIE JOIN MOVIE_COPY
       ON MOVIE.MOVIE_ID = MOVIE_COPY.MOVIE_ID
 WHERE MEDIA_FORMAT = 'D'
 GROUP BY MOVIE_TITLE;
```

93.

```
SELECT CUSTOMER_ACCOUNT_ID, COUNT(*) AS NUM_RENTALS
  FROM CUSTOMER_TRANSACTION A JOIN MOVIE_RENTAL B
       ON A.TRANSACTION_ID = B.TRANSACTION_ID
 GROUP BY CUSTOMER_ACCOUNT_ID;
```

94.

```
SELECT FLOOR(AVG(RETAIL_PRICE_DVD)) AS AVERAGE_DVD_PRICE
  FROM MOVIE;
```

95.

```
SELECT ASCII(';');
```

Pentru Oracle, trebuie adăugată clauza FROM DUAL.

96.

```
SELECT REPLACE(EMPLOYEE_TAX_ID,'-',' ')
       AS EMPLOYEE_TAX_ID
  FROM EMPLOYEE;
```

97.

```
SELECT MOVIE_TITLE,
   CASE
      WHEN RETAIL_PRICE_DVD IS NULL THEN 'None'
      WHEN RETAIL_PRICE_DVD = 0 THEN 'None'
      WHEN RETAIL_PRICE_DVD < 20 THEN 'Bargain'
      WHEN RETAIL_PRICE_DVD < 30 THEN 'Moderate'
      WHEN RETAIL_PRICE_DVD >= 30 THEN 'Premium'
    END AS PRICE_CATEGORY
  FROM MOVIE
 ORDER BY MOVIE_TITLE;
```

98.

```
INSERT INTO MOVIE
  (MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE,
   MOVIE_TITLE)
  VALUES (999, 'Drama', 'NR',
    'How the Database Was Won');
```

99.

```
UPDATE MOVIE
   SET RETAIL_PRICE_DVD = RETAIL_PRICE_DVD + 1
 WHERE RETAIL_PRICE_DVD IS NOT NULL;
```

100.

```
DELETE FROM MOVIE_GENRE
 WHERE MOVIE_GENRE_CODE NOT IN
  (SELECT DISTINCT MOVIE_GENRE_CODE
     FROM MOVIE);
```

Cuvântul cheie DISTINCT este absolut necesar, însă face lista din subinterogare mult mai scurtă, ceea ce determină scăderea eficienței instrucțiunii.

O metodă rapidă și ușoară de învățare a limbajului SQL

Începeți să programați imediat în SQL folosind acest ghid ușor de urmărit. SQL fără mistere explică metodele de a scrie și de a rula interogări în baza de date folosind limbajul SQL. Instrucțiunile pas-cu-pas însoțite de numeroase exemple vă ajută să învățați foarte repede.

Cartea începe cu prezentarea conceptelor bazelor de date relaționale. După aceea, veți învăța să definiți obiectele unei baze de date, să extrageți date folosind limbajul de interogare a datelor (Data Query Language sau DQL), să păstrați datele folosind limbajul de manipulare a datelor (Data Manipulation Language sau DML), să aplicați elemente de control pentru securitatea datelor folosind limbajul pentru controlul datelor (Data Control Language sau DCL), să mențineți integritatea bazei de date, să integrați SQL în aplicații, să adaptați instrucțiunile SQL și multe altele. Incluzând chestionare de verificare la finalul fiecărui capitol și un examen final, această carte vă va învăța să programați în SQL aproape imediat.

ACEST VOLUM UNIC PENTRU AUTOINSTRUIRE OFERĂ

- O metodă simplă de înțelegere a limbajului SQL
- Un chestionar de verificare la sfârșitul fiecărui capitol
- Un examen final la sfârșitul cărții
- Lipsa obligativității cunoașterii unui jargon tehnic
- O abordare în scopul economisirii timpului

Suficient de simplă pentru un începăt
însă suficient de provocatoare pentru
student avansat, cartea SQL fără mis
reprezintă calea dumneavoastră către învățarea detaliată a acestui puternic limbaj de programare.

SQL FARA MISTERE - GHID PENTRU AUTODIDA
184.05 LEI/buc
YATEC2005
Informatica

ISBN 973-7881-07-9
9 789737 881076
www.rosettieducational.ro